DOCUMENT RESUME

ED 163 996 IR 006 791

AUTHOR	White, William W., Ed.
TITLE	Computers and Mathematical Programming. Proceedings of the Bicentennial Conference on Mathematical Programming, November 29-December 1, 1976.
INSTITUTION	National Bureau of Standards (DOC), Washington, D.C.
SPONS AGENCY	Association for Computing Machinery, New York, N.Y. Special Interest Group on Mathematical Programming.; National Bureau of Standards (DOC), Washington, D.C. Applied Mathematics Div.
REPORT NO	NBS-SP-502
PUB DATE	Feb 78
NOTE	381p.; Bicentennial Conference on Mathematical Programming (Gaithersburg, Maryland, November 29-December 1, 1976)
AVAILABLE FROM	Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402 (SD Cat. No. C13.10:502, $5.50)
EDRS PRICE	MF-$0.83 HC-$20.75 Plus Postage.
DESCRIPTORS	*Algorithms; *Computer Oriented Programs; Computers; *Conference Reports; Curriculum Development; Data Bases; Linear Programing; Management Information Systems; *Mathematical Applications; Operations Research; *Programing; Teaching Techniques
IDENTIFIERS	*Computer Software

ABSTRACT

The proceedings of this conference, which examined the relationship between mathematical programming and the computer, contains the texts of most of the 50 papers presented and a summary of one discussion panel addressing such facets of this theme as the design for, use of, implementation of, and implications for mathematical programming software and computations. Particular emphasis was placed on bringing out computer-oriented subject matter not ordinarily presented in a mathematical programming context. (Author/CMV)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED163996

Computers and Mathematical Programming

U S DEPARTMENT OF HEALTH.
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

Proceedings of the Bicentennial Conference
on Mathematical Programming held at the
National Bureau of Standards, Gaithersburg,
Maryland, November 29-December 1, 1976

Edited by

William W. White

IBM Corporation
Poughkeepsie, New York 12602

Sponsored by:

Special Interest Group on Mathematical Programming
Association of Computing Machinery
1133 Avenue of Americas
New York, New York 10036

and

Applied Mathematics Division
Institute for Basic Standards
National Bureau of Standards
Washington, D.C. 20234

acm
sigmap

U.S. DEPARTMENT OF COMMERCE, Juanita M. Kreps, Secretary

Dr. Sidney Harman, Under Secretary
Jordan J. Baruch, Assistant Secretary for Science and Technology

NATIONAL BUREAU OF STANDARDS, Ernest Ambler, Acting Director

Issued February 1978

IR006791

Library of Congress Catalog Card Number: 77-600065

National Bureau of Standards Special Publication 502

Nat. Bur. Stand. (U.S.), Spec. Publ. 502, 383 pages (Feb. 1978)
CODEN: XNBSAV

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1978

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402
(Order by SD Catalog No. C13.10:502). Stock No. 003-003-01893-7 Price $5.50
(Add 25 percent additional for other than U.S. mailing).

ABSTRACT

The Bicentennial Conference on Mathematical Programming, held in Gaithersburg on November 29-December 1, 1976, examined the relationship between mathematical programming and the computer. The more than 50 papers and panel discussions exhibited this theme in terms of the design for, use of, implementation of, and implications for mathematical programming software and computations. Particular emphasis was placed on bringing out computer-oriented subject matter not ordinarily presented in a mathematical programming context. These resulting Proceedings document this Conference, which was jointly sponsored by SIGMAP of the ACM and by the Applied Mathematics Division of the Institute for Basic Standards of NBS.

Keywords: Mathematical programming; linear programming; nonlinear programming; computer software; management science; operations research; computer science; algorithm evaluation; mathematical programming education; software development; databases

BICENTENNIAL CONFERENCE COMMITTEE

GENERAL CHAIRMAN:

Dr. Harvey J. Greenberg, Federal Energy Administration

PROGRAM CHAIRMAN:

Dr. William W. White, IBM Corporation

FINANCE AND REGISTRATION CHAIRMAN:

Professor W. Charles Mylander, III, U. S. Naval Academy

PUBLICITY CHAIRMAN:

Mr. Clarence L. Haverly, Haverly Systems, Inc.

NBS REPRESENTATIVE:

Mr. William Hall, National Bureau of Standards

PROCEEDINGS EDITOR:

Dr. William W. White, IBM Corporation

CONFERENCE ADVISOR:

Professor Saul I. Gass, University of Maryland

PREFACE

First I wish to thank the Bicentennial Conference Committee and our host, The National Bureau of Standards, for their efforts in making this conference so very successful. These proceedings record the presentations, but it is not possible to include the many interesting discussions that took place among the leading researchers as well as newcomers.

Special notice should be taken of subjects not previously considered in conferences on mathematical programming. In particular, two subjects are: (1) interfaces with the computing environment (hardware and software), and (2) database management (including matrix generation/report writing). Bill White is to be congratulated in developing a program that really spans the mainstreams of mathematical programming. Further, he did so by inviting the leaders in these subjects and helping them to form well organized sessions.

Finally, and by no means least, I extend special thanks to our plenary speakers, George Dantzig and William Orchard-Hays. It is certainly appropriate for these founders of the field of Mathematical Programming Systems (MPS) to have provided the foundation for the rest of the program.

Harvey J. Greenberg

FOREWORD

The Bicentennial Conference on Mathematical Programming was held on November 29 - December 1, 1976, at the National Bureau of Standards, Gaithersburg, Maryland. This Conference was jointly sponsored by the Special Interest Group on Mathematical Programming (SIGMAP) of the Association of Computing Machinery, and by the Applied Mathematics Division of the Institute for Basic Standards, National Bureau of Standards, U. S. Department of Commerce.

The basic theme of the Conference was the relationship between mathematical programming and the computer, and the presented papers and panel discussions exhibited this theme in terms of the design for, use of, implementation of, and implications for mathematical programming software and computations. Particular emphasis was placed on bringing out computer-oriented subject matter not ordinarily presented in a mathematical programming context. Both contributed and invited papers were presented, with the contributed papers being refereed: of 54 abstracts received, 47 full papers were submitted, and 30 were selected for presentation by the Session Chairmen, together with the Program Chairman, after the refereeing process.

The Session Chairmen functioned as an extended program committee, and, in consultation with the Program Chairman, really assembled their respective sessions. Special thanks are due them for their contribution to the technical content of the Conference. And thanks go to the referees: G. Bennington, J. P. Blondeau, L. Bodin, B. Buzby, L. Cooper, J. Cord, R. Cottle, R. Davis, R. Dembo, W. Drews, J. Dyer, J. Eddington, F. Fiala, S. Fromovitz, D. Gay, M. Gutterman, M. Harrison, G. Hefley, D. Himmelblau, H. Hoc, R. Jeroslow, D. Klingman, T. Knowles, G. Kochenberger, J. Kowalik, C. Krabek, M. Lenard, R. Marsten, C. McCallum, R. Meyer, M. Minkoff, J. Mulvey, R. O'Neill, T. Prabahakar, L. Pyle, A. Ravindran, H. Salkin, L. Schrage, M. Smith, K. Spielberg, R. Stark, A. Waren, C. White, and J. Whiton.

The Conference Committee did an outstanding job. Under the expert guidance of Harvey Greenberg, the administrative chores were accomplished so as to permit the technical program to be developed with a minimum of impact from secondary sources. In particular, Charles Mylander and Larry Haverly (and his alter ego, Joyce Draper) did yeoman's work on their respective duties. The advice from Saul Gass was especially welcome, particularly when the Conference was in its formative stages.

The National Bureau of Standards was a most appropriate location for this Conference, having been a center of activity in computation and optimization for over a quarter of a century. And, the warm welcome given by B. H. Colvin, Chief of the Applied Mathematics Division, and by A. M. McCoubrey, Director of the Institute for Basic Standards, was reflected as well in the excellent conference facilities and logistics provided by NBS. Thanks and appreciation go to NBS, and especially to Bill Hall, the NBS representative, and to Sarah Torrence, of the NBS staff.

William W. White

7

TABLE OF CONTENTS

REMARKS ON THE OCCASION OF THE BICENTENNIAL CONFERENCE ON MATHEMATICAL PROGRAMMING: THE EARLY ROLE OF N.B.S.
G. B. Dantzig, Stanford University 1

KEYNOTE ADDRESS: Energy Models and Large-Scale Systems Optimization
G. B. Dantzig and S. C. Parikh, Stanford University 4

PLENARY ADDRESS: The Challenge of Analytic Use of Computers for Global Problems
W. Orchard-Hays, IIASA 11

RECENT ALGORITHMIC ADVANCES - I
Organized and Chaired by A. Orden, University of Chicago

Implementation and Application of a Nested Decomposition Algorithm
J. K. Ho, Brookhaven National Laboratory 21
A Stepping-Stone Parallel-Cut Method for Integer Programming
T. Cheung, University of Ottawa 31
Monomial Programming
T. L. Shaftel, University of Arizona and Queen's University
G. L. Thompson, Carnegie-Mellon University
Y. Smeers, Catholic University of Louvain 38
Implementation and Use of Nonlinear Cost Multicommodity Flow Subroutines
H. H. Hoc, Ecole Polytechnique 51
Implicit Representation of Triangularity Constraints in Linear Programming
G. Gunawardane, University of Sri Lanka and University of Chicago
L. Schrage, University of Chicago 59
An Efficient General Algorithm for the Computation of Linear Decision Rules
S. F. Thomas, Caribbean Industrial Research Institute 65

APPLICATIONS: PUBLIC SECTOR
Organized by M. Held, IBM Systems Research Institute, and Chaired by H. Crowder, IBM Research

*The Structure and Solution Techniques of the Project Independence Evaluation System
F. Murphy, Federal Energy Administration 73
National and Interregional Programming Models of Land and Water Use and the Environment
E. O. Heady, K. J. Nicol and D. Dvoskin, Iowa State University 86
A Nonlinear Programming Approach to Preference Maximized Menu Plans
J. L. Balintfy and P. Sinha, University of Massachusetts 97

*Denotes Invited Paper

ISSUES IN THE EVALUATION OF MATHEMATICAL PROGRAMMING ALGORITHMS

Organized and Chaired by R. H. F. Jackson, National Bureau of Standards, Boulder
Sponsored by the Working Committee on Algorithms of the Mathematical Programming Society

*On the Analysis and Comparison of Mathematical Programming Algorithms and Software
R. S. Dembo, Yale University
J. M. Mulvey, Harvard University 106
*Avenues for Research in the Evaluation of Mathematical Programming Algorithms
H. J. Greenberg, Federal Energy Administration
R. P. O'Neill, Louisiana State University . . . . (Presentation Only)
The Evaluation of Unconstrained Optimization Routines
L. Nazareth, Argonne National Laboratory
F. Schlick, University of Illinois 117

PANEL ON ISSUES IN THE EVALUATION OF MATHEMATICAL PROGRAMMING ALGORITHMS

Organized and Chaired by R. H. F. Jackson, National Bureau of Standards, Boulder
Sponsored by the Working Committee on Algorithms of the Mathematical Programming Society

R. S. Dembo, Yale University
J. J. Filliben, National Bureau of Standards
H. J. Greenberg, Federal Energy Administration
J. L. Kreuser, World Bank
R. P. O'Neill, Louisiana State University 134

APPLICATIONS: PRIVATE SECTOR

Organized and Chaired by D. Soults, Management Science Systems, Alexandria

*Large Scale Mathematical Programming: A Total System Approach
T. Prabhakar, Union Carbide, South Charleston 143
A Search Enumeration Algorithm for a Multiplant Multiproduct Scheduling Algorithm
S. Morito and H. M. Salkin, Case Western Reserve University 144

DATABASE MANAGEMENT SUPPORT

Organized and Chaired by H. Patton, Exxon

*An IMS-Gamma 3 Database Editor
E. B. Brunner, Gulf Oil, Pittsburgh 152
Software Tools for Combining Linear Programming with Econometric Models
M. J. Harrison, National Bureau of Econometric Research 165
Database Management Techniques for Mathematical Programming
R. H. Bonczek, C. W. Holsapple and A. B. Whinston, Purdue University 171

PANEL ON THE IMPLICATIONS OF THE HARDWARE ENVIRONMENT

Organized and Chaired by J. Tomlin, Stanford University

E. Hellerman, Census Bureau
D. Stevenson, Institute for Advanced Computation, Sunnyvale . . . . (Discussion Only)

*Denotes Invited Paper

NUMERICAL METHODS

Organized and Chaired by R. Bartels, The Jonns Hopkins University

*Convergence of the Diagonalized Method of Multipliers
R. H. Byrd, The Johns Hopkins University 180
*Direct Approaches for the Minimax Problem
A. R. Conn, University of Waterloo 184
*Numerical Aspects of Trajectory Algorithms for Nonlinearly Constrained Optimization
W. Murray, National Physical Laboratory
M. H. Wright, Stanford Universit 194
Optimization Algorithms Derived from Nonquadratic Models
J. S. Kowalik, Washington State University 205

RECENT ALGORITHMIC ADVANCES - II

Organized by A. Orden, University of Chicago, and Chaired by S. I. Gass, University of Maryland

Algorithms for a Class of 'Convex' Nonlinear Integer Programs
R. R. Meyer and M. L. Smith, University of Wisconsin 210
Extreme Point Ranking Algorithms: A Computational Survey
P. G. McKeown, University of Georgia 216
A New Alternating Basis Algorithm for Semi-Assignment Networks
R. Barr, Southern Methodist University
F. Glover, University of Colorado
D. Klingman, University of Texas at Austin 223
Recent Developments in Vehicle Routing
B. L. Golden, Massachusetts Institute of Technology 233
Balasian-Based Enumeration Procedures: A Study in Computational Efficiency
J. H. Patterson, Purdue University 241

SOLUTION STRATEGIES AND TACTICS

Organized and Chaired by J. Kalan, University of Texas at Austin

A Study of the Effect of LP Parameters on Algorithm Performance
C. H. Layman and R. P. O'Neill, Louisiana State University 251
Sensitivity Analysis For Parametric Nonlinear Programming Using Penalty Methods
R. L. Armacost, U. S. Coast Guard
A. V. Fiacco, The George Washington University 261
*Approximations in Linear Programming
J. Kalan, University of Texas at Austin .... (Presentation Only)
*The Generalized Inverse in Nonlinear Programming -- Equivalence of the Kuhn-Tucker, Rosen and Generalized Simplex Necessary Conditions
L. D. Pyle, University of Houston 270

*Denotes Invited Paper

MATHEMATICAL PROGRAMMING EDUCATION

Organized and Chaired by J. F. Shapiro, Massachusetts Institute of Technology

*Teaching Mathematical Programming to the Consumer
M. L. Fisher, University of Pennsylvania 275
On Teaching Linear Programming Fundamentals
J. M. Mulvey and R. D. Shapiro, Harvard University 279
Experiments with Computer Aided Self Paced Instruction for Mathematical Programming Education
A. Ravindran, A. Sinensky and T. Ho, Purdue University 286
Importance of Modelling for Interpretation of Linear Programming Models
L. W. Swanson, Northwestern University 294
Interactive Computer Codes for Mathematical Programming Education
R. P. Davis and J. W. Chrissis, Virginia Polytechnic Institute and State University 302

PANEL ON EXECUTOR/SUPERVISOR SUBSYSTEMS AND THE SOFTWARE ENVIRONMENT

Organized and Chaired by E. Hellerman, Census Bureau

D. Carstens, Burroughs Corp.
J. Kalan, University of Texas at Austin
C. B. Krabek, Control Data Corporation
W. Orchard-Hays, IIASA . . . . (Discussion Only)

PROBLEM SOLVING SYSTEMS: CAPABILITIES AND STRUCTURE

Organized and Chaired by J. L. Nazareth, Argonne National Laboratory

*A Problem Solving System for Nonlinear Least Squares
B. A. Arnoldy, Argonne National Laboratory
K. Brown, University of Minnesota 310
*An Iteratively Reweighted Least Squares System
V. Klema, National Bureau of Economic Research 319
An Experimental Interactive System for Integer Programming
M. Guignard, University of Pennsylvania
K. Spielberg, IBM Corp., White Plains 328
An Accelerated Technique for Ridge Following Using Conjugate Directions
E. H. Neave, Queen's University
T. L. Shaftel, University of Arizona and Queen's University 338

PANEL ON SELECTION AND EVALUATION

Organized and Chaired by A. C. Williams, Mobil Oil, Princeton

J. G. Colahan, Pennwalt Corp., Philadelphia
J. R. Ellison, Mobil Oil, Beaumont
M. M. Gutterman, Standard Oil of Indiana, Chicago
D. S. Hirshfeld, Management Science Systems, Falls Church 354

*Denotes Invited Paper

SOFTWARE DEVELOPMENT

Organized by G. T. Martin, Control Data Corp., New York, and Chaired by C. B. Krabek, Control Data Corp., Minneapolis

*Development of Mathematical Programming Systems
D. S. Hirshfeld, Management Science Systems, Falls Church . . . . (Presentation Only)
*Experiences in the Development of a Large Scale Linear Programming System
R. Sjoquist, Control Data Corporation, St. Paul 358
*Experiences in Developing OMNI
C. L. Haverly, Haverly Systems Inc. . . . . (Presentation Only)
*Experiences in the Development of PDS/MAGEN
S. Halliburton, Haverly Systems Inc. . . . . (Presentation Only)

OPERATIONAL MANAGEMENT

Organized and Chaired by J. R. Estabrook, Union Carbide, Bound Brook

*Math Programming Users vs. the Computer Center (A Personal Perspective As Seen From A Foxhole)
J. R. Ellison, Mobil Oil, Beaumont 362
*Operational Management of Mathematical Programming Based Planning Systems
E. G. Kammerer, Union Carbide, South Charleston 364
*Managing a Large Scale Production and Distribution Scheduling System
K. Goldfisher, Nabisco Inc 368

* Denotes Invited Paper

REMARKS ON THE OCCASION OF THE
BICENTENNIAL CONFERENCE ON MATHEMATICAL PROGRAMMING
THE EARLY ROLE OF N.B.S.

George B. Dantzig

Stanford University

It's a great pleasure to be here today. I'm very glad that Dr. McCoubrey, the Director of the Institute for Basic Standards of the National Bureau of Standards, in his introductory remarks, told us for whom NBS worked, namely the consumer, industry, the scientific community, and educational institutions. It is nice to learn that some parts of the government truly work for us because some of us had come to believe that it was the other way around.

Dr. McCoubrey mentioned the early days of linear programming and the cooperative role played by the Bureau with Air Force Project SCOOP. I would like to make this my theme today.

As some of you may recall, I was the mathematical advisor to the Air Force Comptroller at the time that linear programming was born. Thinking back to the early days, I have discovered a remarkable coincidence that I would like to pass on to you between the date chosen for this commemorative conference, November 29, 1976, and the date when linear programming began. Preliminary flirtations with the idea started in the fall of 1946. By November 29, 1946, exactly 30 years ago, linear programming was conceived.

During the war I took part in the planning activities of the Air Force. In the immediate postwar period, I was in the throes of trying to decide whether to stay with government, begin an academic career, or do research for industry. I couldn't make up my mind. There were some people in the Air Force (particularly Dal Hitchcock and Marshall Wood) who were very keen on having me stay. As bait they suggested that I try to the mechanize of the planning process. This challenge intrigued me. In the fall of '46, I toyed with different approaches. By November 29, 1946, the idea came that perhaps the Input-Output technique of Leontief (for which he received the Nobel Prize in 1973) could be suitably generalized. In the winter of '46, my work began in earnest; by June of '47, the linear programming model as we know it today was well along.

Our research from early 1947 on was influenced by a conference arranged by Aiken at Harvard. It was here that Marshall Wood and I became first exposed to the idea of an electronic computer. It was a wonderful conference. Although it was only a gleam in the eyes of the speakers, they talked about electronic computers as if they really existed and indeed with capabilities very much as they have today. I was overwhelmed with the potential of this new tool. To appreciate what happened in the early stages of linear programming, it's well to remember that we believed that computers would become practical within a year or two. We acted accordingly.

It would be interesting to speculate how many important developments might never have happened if one knew that it would take almost two decades for powerful computers to become a practical reality.

From the beginning, it was this gleam in the eye of the designers that fast, practical computers really would soon become available that motivated computational development of linear programming. It resulted in the Air Force decision to mechanize the planning process. The name for the effort was Air Force Project SCOOP, standing for Scientific Computation of Optimum Programs. The Office of Naval Research, particularly Dr. Mina Reese, who is well known to many of you, played an important role. She introduced us to John Curtiss and his

---

Edited from a transcription taken at the Conference. For additional background information and more detailed references on SCOOP and early mathematical programming influences and activities, see Chapter 2 of G. B. Dantzig, LINEAR PROGRAMMING AND EXTENSIONS, Princeton University Press, 1963.

mathematics group at the Bureau. Her office subsidized related research.

Our group was not technically equipped to supervise or evaluate the development of computers. We turned to the Bureau of Standards to serve as our technical agents, to keep us informed about computer developments. And so it came about that I became one of the behind-the-scenes sponsors of the early development of computers. The Air Force Comptroller transferred huge sums of money to the NBS for this purpose. There were close contacts with Sam Alexander whose group at NBS built the SEAC. Air Force money helped NBS fund the building of BINAC, UNIVAC, and also some IBM component research. I don't claim that SCOOP was the only sponsor (directly or indirectly) in this field; there were others, for example the Bureau of the Census; nor did SCOOP sponsor the building of SWAC under Harry Huskey at the Bureau's Institute of Numerical Analysis at UCLA.

I am particularly grateful for the advice of Albert Cahn who worked for Curtiss. He recommended two key people whom he said I should consult regarding the relation of linear programming to economics, mathematics, and numerical analysis: the first was the economist, Tjalling Koopmans; the other, the famous mathematician, Johnny Von Neumann. In June of 1947, I went to the Cowles Foundation at the University of Chicago to see Koopmans. This contact initiated his interest and soon the interest of other young economists (many now well known) in the relationship between mathematical programming and economics. In 1975 Koopmans received the Nobel Prize for his contributions to the theory of resource allocation. In the fall of 1947, I went with Curtiss to Princeton to see Von Neumann at the Institute of Advanced Study. In the course of our discusssions, Von Neumann stated the duality theorem, related it to game theory, and made other observations that laid the mathematical foundations. (The history of the duality concept makes an interesting story in itself.)

In June of 1948, John Curtiss again introduced me, this time to his brother-in-law, Al Tucker at Princeton. Soon thereafter Tucker with his students Harold Kuhn and David Gale started a seminar which resulted in their well known contributions to duality theory, game theory and nonlinear programming. Von Neuman and Tucker spearheaded the interest of mathematicians.

While this academic interest was growing, work began in earnest within the Air Force to mechanize the planning process. During the period 1947-52 there were two main branches to our efforts -- one practical, the other theoretical. The goal of the practical branch was to implement quickly and to solve routinely very, very large dynamic programs required for planning. These systems were so large that, even by today's standards, they would be beyond anyone's capability to optimize as a single linear program. For practical planning Marshall Wood and I invented a hierarchical stepwise optimization scheme called a "triangular model". We echelonned the activities of the Air Force in "parasitic" order -- namely on the top rung were the combat units (which took from everybody but gave nothing in return); next below them were the support units in the combat zone (which took from everybody below them but gave nothing in return); and so forth down to support units that recruited and trained personnel and the units which bought and distributed supplies. Although called a triangular model it could be applied to general matrices by first making a triangular approximation and then making iterative corrections by adding on additional lower rungs in the hierarchical ordering. Murray Geisler's group formulated Air Force planning problems in this manner. Sted Nobel and others extended the formulation to sectors of the national economy.

The triangular model scheme was implemented computationally in the spring of 1949 by Mike Montalbano of the National Bureau of Standards. On the plane coming to this conference I ran into Mike. We reminisced about his achievement. Mike did a very remarkable thing: he used IBM punched card equipment, the only equipment available, in a way that no one had ever used it before. To iteratively carry out the programmed steps, he processed the cards through a sequence of machines: tabulators, sorters, reproducers, collaters, and IBM 604's, all arranged in a great big circle. Because the equipment was unreliable, he also processed the cards through two pieces of similar equipment serially in order to have one machine check the computations of the other.

One day during the development of the computational system by Montalbano, Wassily Leontief visited Washington. I asked Wassily if he would like to come on over to the Bureau of Standards to take a look (they had a downtown office where tests were being run). It was a Sunday, and Mike was pleased to put on a demonstration of just how wonderful his system was. He pointed out that the equipment was unreliable and how he had to get after IBM regularly to make it work. He pointed out the features of his system for checking errors. "For example", he said, "See those two reproducers, one following the other? If the holes punched by the second do not match exactly those punched by the first, the second reproducer will stop, turn on red lights, and we will know that the last card processed is wrong."

As the cards were running through the

second reproducer, Mike said, "You see, at this point everything is working fine, there are no red lights."

But on the side to me he said, "Isn't it making a loud crunching noise?"

As he pulled the cards from the output hopper, we saw that the columns were laced full of holes -- indeed they were nothing but confetti. The cards were clearly overpunched, but no red lights had turned on. Leontief tapped me on the shoulder and said, with his Russian accent, "There, there, I understand these things. Don't vorry. A machine is like a voman, very temperamental." This story is a classic example of System Antics -- the science of why complex systems work poorly if at all. Something unforeseen always happens. In this case someone the night before had changed the standard positions of the control brushes on the reproducer and forgot to change them back.

Turning now to the theoretical branch, namely research on the linear programming model, here also the Air Force turned to the Bureau for help. Jack Laderman and his group at the Mathematical Tables Project in New York, using hand calculators, solved Stigler's nutrition problem using the Simplex Method (December 1948). This was the first real test of the method. To do the computations, Laderman handed out small worksheets. These completed worksheets were pasted together to form a huge "table cloth". In going through some old correspondence recently, I found a letter from Oskar Morgenstern -- he wanted to come down to Washington to see the famous table cloth. I wonder whatever happened to it?

In early 1950, Montalbano with Corky Diehm wrote the first Simplex Code and made successful runs with it on the SEAC computer.

There were others at the Bureau well known to you who contributed in a major way to the early development. Alex Orden, who is here today, was with the Bureau for a brief period before he joined our Air Force group. There was Ted Motzkin, Alan Hoffman, and Leon Gainen. George Suzuki, now with the Bureau, was with us at SCOOP, as was Joe Natrella, whose wife Mary is still at the Bureau. During the early 1950's, Hoffman, Mannos, Sokolowsky and Wiegman ran comparative tests of the Simplex Method with alternative algorithms. At I.N.A., Motzkin, Raiffa, Thompson, and Thrall developed the Double Description Method; Motzkin and Shoenberg developed the Relaxation Method; and C. B. Tompkins, the Projection Method.

Those were the days to remember! I, personally, am grateful to the Bureau for (1) their pioneering the construction of computers and their sponsorship of others in the field; (2) their design of the first linear programming software; (3) their contacts with economists and mathematicians which brought about early interest by the academic community; (4) their sponsorship of symposia on mathematical programming; (5) their research on mathematical programming theory, algorithms, and their making comparative tests. These contributions played a major role in the early rapid development of the mathematical programming field.

ENERGY MODELS AND LARGE-SCALE SYSTEMS OPTIMIZATION*

George B. Dantzig and Shailendra C. Parikh
Stanford University

The optimization of large-scale dynamic systems represents a central area of research whose successful outcome could make important contributions to the analysis of crucial national and world problems. Although a great number of papers have been published on the theory of solving large-scale systems, not much in software exists that can successfully solve such systems. We believe that there has been little progress, because there has been little in the way of extensive experimentation comparing methods under laboratory-like conditions on representative models. At Stanford's Systems Optimization Laboratory (SOL), to bridge this gap between theory and application, we

(1) develop experimental software for solving large-scale dynamic systems,
(2) systematically compare proposed techniques on representative models,
(3) record and disseminate information regarding experimental results.

PILOT Energy-Economic Model

Dynamic models that describe the interactions between the energy sector and the general economy help in providing a focus to our research in experimenting with large-scale optimization models. Models of this type are under development by a number of groups to study the energy crisis and a probable future crisis in the area of food (agriculture). Developers of these models could make effective use of techniques for solving large-scale systems, _if_ they were available.

Today's policy makers at industrial, governmental and international levels are faced with the decisions on providing the needed energy in the years to come at acceptable social cost. Such decisions must take into account many complex interactions related to the technology of energy supply, environmental side effects, energy resource conservation, etc., as well as the national welfare considerations of unemployment, inflation and living standards.

Some of the important questions that must be considered in detail in the formulation of the energy policy (or policies) are the following:

(1) Are we using up our cheap energy resources too quickly?
(2) Are we making sufficient investments now so that new energy technologies will come into commercial operation when needed in future years?
(3) Do we have sufficient physical capacity to build the required new plant and equipment in the energy and nonenergy sectors without seriously hampering growth in consumer consumption, or will some sacrifice in consumption be necessary?
(4) What are the various energy options under different patterns of crude oil import price realizations?
(5) What will the short and long term impact be if oil and gas discoveries are less than predicted?
(6) Can we find an energy policy that is robust, i.e., one which hedges against various contingencies?

It is our belief that dynamic mathematical programming models can, at the very least, provide analysis and information on these and other questions and can substantially improve the understanding of the interactions that must be considered. Such models have been developed at IIASA, at the Electricité de France and by various groups in the United States.

In the Systems Optimization Laboratory at Stanford we have under development a linear programming model for assessing energy-economic options in the United States, called PILOT [7]. It spans a wide spectrum of activities of the economy, from exploration and extraction of raw energy to industrial production and consumer demands for all goods and services. The data requirements therefore cut across many different sources--consumer surveys, import/export and trade balance data, manufacturers surveys, mining data, input/output and capital coefficients, energy consumption and substitution data, energy technology data from Brookhaven National Laboratory and oil and gas exploration and production data. Hence, there is a nontrivial problem of achieving consistency and of selecting a meaningful level of detail so that the model stands as a whole rather than as a conglomeration of parts that could collapse under careful analysis. We believe that the initial version of the model being built will meet this test.

---

*Research of this paper was partially supported by the Office of Naval Research Contracts N00014-75-C-0267, N00014-75-C-0865; U.S. Energy Research and Development Administration Contract E(04-3)-326 PA #18; the National Science Foundation Grants MCS71-03341 A04, DCR75-04544; the Electric Power Research Institute Contract RP 652-1; and the Stanford Institute for Energy Studies at Stanford University.

The PILOT model is a statement in physical flow terms, to the extent possible, of the broad technological interactions within and across the sectors of the economy, including, but in greater detail, the energy sector. A typical run of the model describes what the country could achieve in physical terms over the long term, say 40 years.

A preliminary version of the model has been completed and several useful scenarios have been run [16]. In 1977, improved versions of the model will be developed, with more detail regarding exploration and extraction by regions, more detailed modeling of various conversion processes, better representation of foreign trade, substitution, financial flows and the effect of prices on demand and production.

The initial version of the PILOT model is an eight period, 40 year model which has approximately 800 constraints and 2000 variables.

The model is a description in input/output terms of the industrial processes of the economy and the demands for consumption, capacity formation, government services and net exports. The description of the processes that provide useful energy to the economy constitutes the detailed energy submodel. This consists of technological input/output descriptions of the raw energy extraction and the energy conversion processes as well as the energy import and export activities. Four linkages interconnect the energy sector and the rest of the economy: energy demands of the economy, bill of goods needed for energy processing and capacity expansion, total manpower available to all sectors (including energy) and a trade balance constraint which requires the equating of total exports to total imports when these items are evaluated in 1967 dollars over successive five year periods. See Figure 1, which shows the major blocks of coefficients in a time period, and its link to the next time period show below and to the left of the dotted lines.

As noted earlier, the equations of the model express the balances of various physical flows. For the energy sector, the balances of coal, oil, gas, etc. are each expressed in BTU units. For the economy, the units are 1967 dollars, which are obtained by weighting the underlying physical flows of goods and services (assumed to be in fixed proportions) by 1967 prices per unit.

The industrial sectors of the economy are represented by a 23-order input/output matrix. The sectors are grouped as follows: 5 energy sectors, 1 agriculture, 1 nonenergy mining, 5 energy intensive manufacturing, 4 energy nonintensive manufacturing, 4 services and 3 capital formation.

Consumption is modeled in terms of the consumption vector of the average consumer. This sector does not have a fixed bill of goods per capita but varies as a function of a parameter representing the real consumption income attained per capita. Based on an analysis of historical data, consumption of any item as function of average consumption income is nearly linear [2].

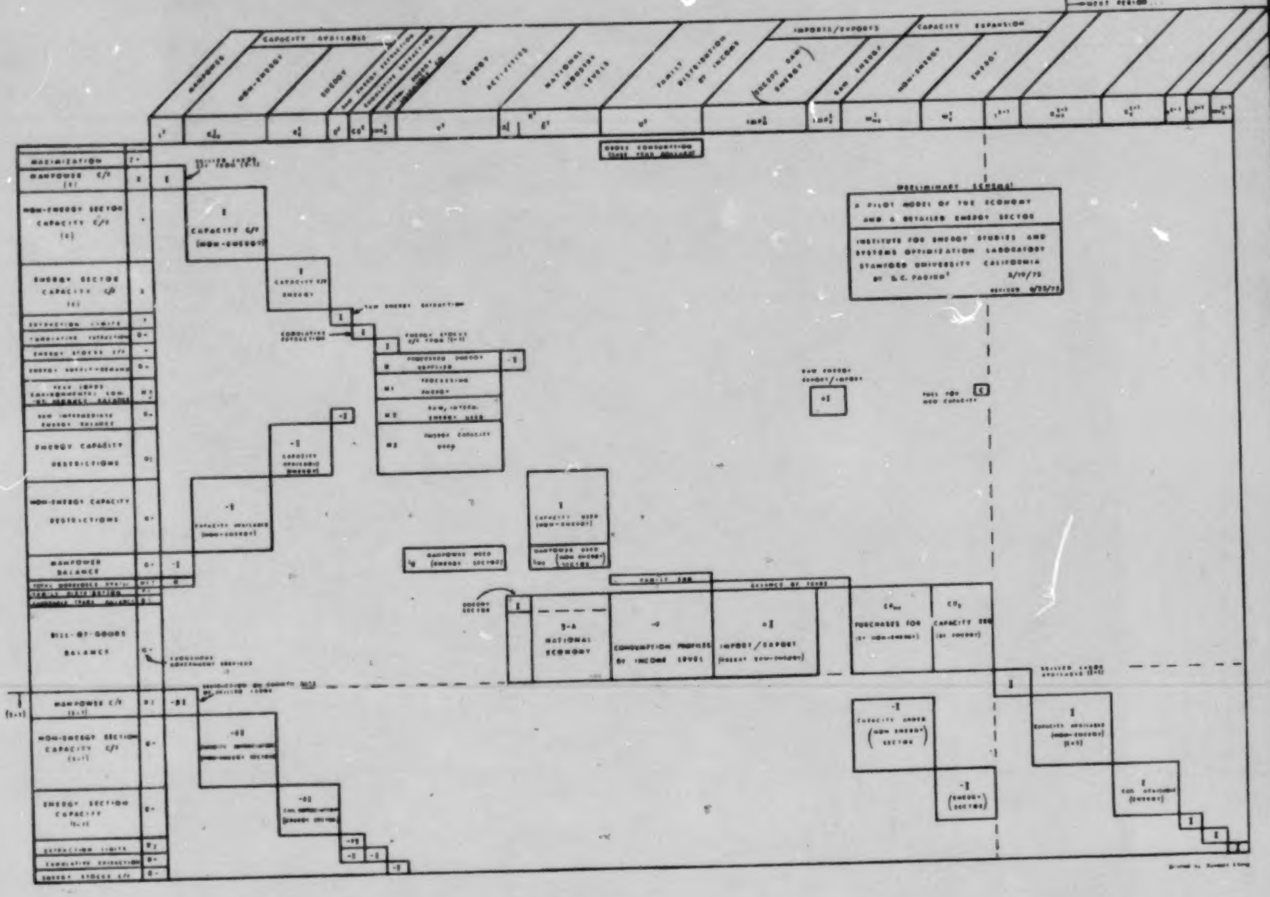
Capacities for each of the 18 nonenergy sectors and all of the energy processes are differentiated from one another. The heterogeneous capital equipment of the nonenergy sectors is depreciated, whereas the energy facility capacities are assumed instead to have undepreciated, fixed lifetimes. Construction lags are used to specify the time it takes to build new capacity. These construction lags may be chosen individually for all 18 nonenergy sectors as well as for all energy facilities.

The exports are treated as final demand items. The imports are considered in two parts, competitive and noncompetitive. The noncompetitive imports are for those goods and services for which no domestic substitutes exist. They are treated as a part of the technology of the consuming industrial sector. On the other hand, competitive imports of goods and services for which domestic substitutes do exist are treated as activities that can augment the domestic production. Finally, a trade balance constraint ties together the amounts of all imports and exports. Typically, we have assumed over a five year period that the value of total exports be matched by that of imports.

The labor force is assumed to be exogenously given. Also, average labor productivity is assumed to grow at an exogenously given rate. In sample runs, the "standard of living" attained appears to be very sensitive to this factor. In the base case, 2% per year productivity growth is assumed.

The detailed energy sector contains the conventional energy technologies, such as oil refinery, coal fired plant, etc., as well as new technologies, such as coal synthetics, oil shale, plutonium fired reactors, etc.

The description of the energy sector includes an accounting of reserves remaining of three exhaustible energy resources: oil, gas and uranium. For oil and gas, finding-rate functions are used to specify the amount of oil in place and gas reserves to be found for a given amount of drilling effort. The level of drilling effort is endogenously determined. The advanced (and expansive) techniques of secondary and tertiary recovery are also defined in the model. Alaskan oil production and the Trans-Alaskan Pipeline System (TAPS) construction are assumed to be exogenously given. For natural uranium, increasing facilities and manpower are required to extract a ton of ore as more and more is extracted. In particular, progressively higher amounts of uranium mining and milling capacity are needed to process the poorer grade ore per pound of uranium oxide obtained. While, in principle, generalized linear programming could be used to model the nonlinear functions, we, in fact, replaced the nonlinear functions by broken line fits.


CAPACITY AVAILABLE
MANPOWER
NON-ENERGY
ENERGY
ENERGY ACTIVITIES
NATIONAL INDUSTRY LEVELS
FAMILY DISTRIBUTION BY INCOME
IMPORTS/EXPORTS
CAPACITY EXPANSION
NON-ENERGY
ENERGY
NEXT PERIOD...
GOODS CONSUMPTION
MAXIMIZATION
MANPOWER C/F
NON-ENERGY SECTOR CAPACITY C/F
ENERGY SECTOR CAPACITY C/F
EXTRACTION LIMITS
CUMULATIVE EXTRACTION
ENERGY STOCKS C/F
ENERGY SUPPLY-DEMAND
RAW INTERMEDIATE ENERGY BALANCE
ENERGY CAPACITY RESTRICTIONS
NON-ENERGY CAPACITY RESTRICTIONS
MANPOWER BALANCE
BILL-OF-GOODS BALANCE
EXOGENOUS GOVERNMENT SERVICES
I
CAPACITY C/F (NON-ENERGY)
CAPACITY C/F ENERGY
RAW ENERGY EXTRACTION
CUMULATIVE EXTRACTION
ENERGY STOCKS C/F FROM (t-1)
PROCESSING ENERGY
ENERGY CAPACITY USED
-I
CAPACITY AVAILABLE (ENERGY)
CAPACITY AVAILABLE (NON-ENERGY)
CAPACITY USED (NON-ENERGY)
MANPOWER USED (ENERGY SECTOR)
MANPOWER USED (NON-ENERGY SECTOR)
RAW ENERGY EXPORT/IMPORT
FUEL FOR NEW CAPACITY
ENERGY SECTOR
FAMILY USE
BALANCE OF TRADE
I-A NATIONAL ECONOMY
-C
CONSUMPTION PROFILES BY INCOME LEVEL
±I
IMPORT/EXPORT (EXCEPT RAW-ENERGY)
PURCHASES FOR (NON-ENERGY)
CAPACITY USE (ENERGY)
(t+1)
MANPOWER C/F (t+1)
NON-ENERGY SECTOR CAPACITY C/F (t+1)
ENERGY SECTOR CAPACITY (t+1)
EXTRACTION LIMITS
CUMULATIVE EXTRACTION
ENERGY STOCKS C/F
-I
CAPACITY UNDER (NON-ENERGY SECTOR)
(ENERGY SECTOR)
CAPACITY AVAILABLE (NON-ENERGY) (t+1)
CAP. AVAILABLE (ENERGY)
PRELIMINARY SCHEMA
A PILOT MODEL OF THE ECONOMY AND A DETAILED ENERGY SECTOR
INSTITUTE FOR ENERGY STUDIES AND SYSTEMS OPTIMIZATION LABORATORY STANFORD UNIVERSITY CALIFORNIA
BY S.C. PARIKH
3/19/75
REVISED 4/28/75

The PILOT model can be used in conjunction with any desired social objective. Consideration of reasonable alternative objectives requires further investigation. Some objectives may require their expression partly in the form of extra constraints as well as in the values of the coefficients of the maximand. The objective chosen in the base case in the initial version of the model is the undiscounted sum of the gross national consumption over all 40 years, subject to (1) a "monotonic per capita" constraint which states that the average per capita consumption must be nondecreasing over time, and (2) an initial condition which sets a lower limit on first period consumption. Experimentation with other maximands is possible. For example, we have experimented with discounted gross national consumption.

The objective of PILOT is designed to permit one to determine feasible solutions to our economy--in particular, what level of investment (in physical terms) both in the energy and non-energy sectors is necessary in order to have as high a standard of living as possible for the growing population.

Once the physical flows are determined, it is possible to solve a related financial investment model. The financial flow model calculates a system of prices, taxes, salaries, profits, interest rates, etc. that is internally consistent in the sense that all economic agents--consumers, producers, government, etc.,--receive sufficient monies to pay their expenses for the specified physical flows. The prices generated by the model can be adjusted to be, at the same time, noninflationary, i.e., to have the same buying power as base year prices. We also are giving some thought to incorporating in the model production and demand functions to adjust input/output coefficients as a function of prices. In the initial version of the model these coefficients are fixed, however.

To illustrate some of the output of the model, a typical base case assumes a 2% growth in labor productivity, 20% limit on the total amount of energy purchased overseas, certain limits on the rate of growth of coal production, etc. A base case run computes consumption income (income after taxes and savings). In 1975 this income per capita (in 1975 dollars) was about $4500. Based on these assumptions, the model states it is possible for the country to have a future consumption income per capita relative to consumption per capita in 1975 as follows:

1975	1980	1985	1990	1995	2000	2005	2010
1.0	1.0	1.2	1.6	1.7	1.8	2.0	2.2

This possible future can be compared with that obtained from another scenario, which is the same as the base case but restricts the use of nuclear power plants. This naturally results in a lower achievable per capita consumption income. Relative to 1975, the results are as follows:

1975	1980	1985	1990	1995	2000	2005	2010
1.0	1.0	1.2	1.5	1.5	1.5	1.5	1.5

Comparing the two scenarios at the year 2010, we have 2.2 vs. 1.5, i.e., the nuclear restriction could reduce the "standard of living" achievable by 2010 by 30%. This conclusion has been criticized because the model assumes that consumption patterns of people at different income levels will remain unchanged (i.e. they won't practice conservation or change their life styles) and that production methods will be no more efficient in the use of energy in the future than they are today. This criticism we feel has merit and we are, accordingly, considering revisions in the model to include more substitution and conservation possibilities.

Because of excellent liaison with other groups working in the energy field, we anticipate that the proposed physical flow model will contribute to the formulation and solution of the more detailed specialized models under development elsewhere. In particular, the PILOT model is one being compared with other models by the newly formed U.S. Energy Modeling Forum in its examination of the feedback effects from the energy sector on the economic growth.

Solving Multi-Time-Period Models*

Solving energy models by commercial linear programming software is proving to be expensive. Large-scale techniques, such as those under development at the Systems Optimization Laboratory, are currently under test to see if they can help solve these models more efficiently.

Conceptually, the decomposition principle [8] has proved to be a natural approach to breaking up large systems and to decentralized decision making. So far, computational experience has been limited, but it is known that several devices can be effectively combined with the decomposition principle to accelerate the iterative process. Classic research along these lines can be found in the work of Rosen [17], Beale [3], Gass [9], Bell [4], Abadie [1], Bennett [5], as well as in the joint work of Wolfe and Dantzig [8]. Areas of SOL research include (1) intertemporal models with staircase structures, (2) the continuous simplex method for linear control problems with state-space constraints, and (3) general large-scale dynamic nonlinear problems.

Recently, experiments have been conducted at SOL comparing the Decomposition Principle approach with a special variant of the simplex method known as Generalized GUB. These tests were limited in nature but indicated that Generalized GUB is superior. Our research on dynamic systems is therefore examining variants of the simplex method as well as special methods for decoupling staircase and block-triangular systems. See Figure 2. We plan to compare these approaches with the nested decomposition algorithm of James K. Ho and Alan S. Manne for staircase systems [12].

Staircase systems have historically proven to be very difficult, usually requiring a disproportionately large number of simplex iterations to solve.

---

* This section is based on a summary prepared by J. A. Tomlin.

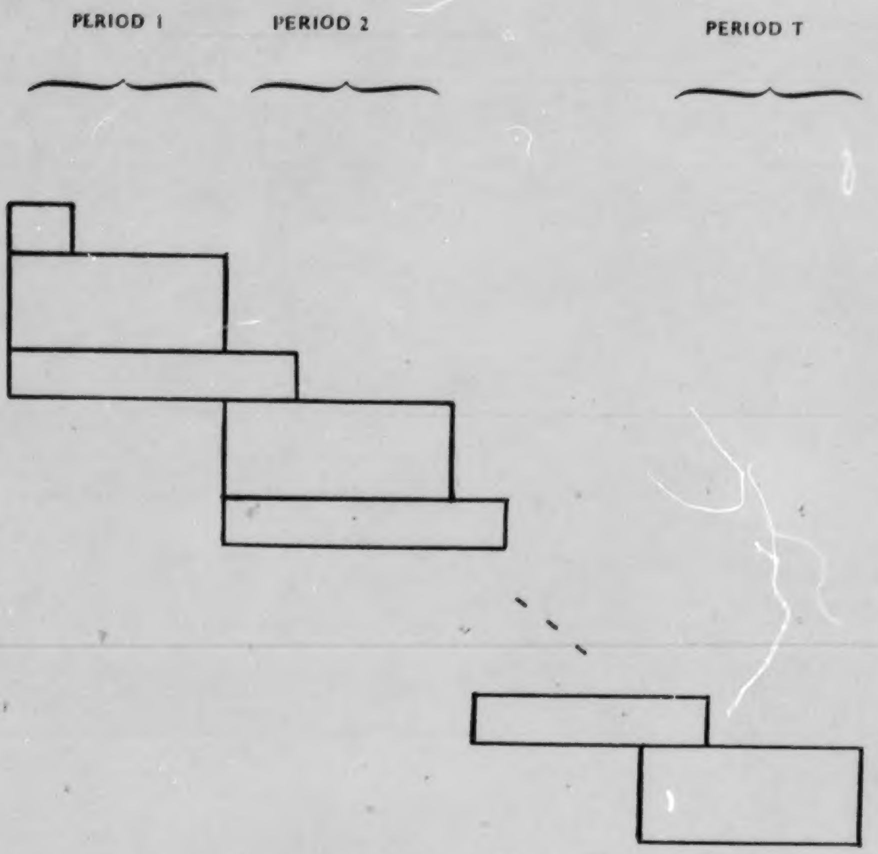


FIGURE 2. The Staircase Structure of the PILOT Energy Model

J.A. Tomlin of SOL has experimented with a partial decoupling of time periods within a model by relaxing the intertemporal constraints. The expectation is that such a relaxation will result in a more easily solvable model, whose solution can be used as a starting point for the real model, using some "gradual" approach to restore the intertemporal constraints. As might be expected, the results of this approach are quite problem dependent, and sensitive to the degree and the kind of relaxation employed. It appears that tightly constrained economic planning models, of the type available to us for these experiments, require a more sophisticated approach. Other types of staircase models are being acquired to further test this idea.

One of the more promising methods that have been investigated for reducing solution time for dynamic models involves several modifications to the simplex method designed to take advantage of the special properties and behavior of such models. The essential property of interest is the tendency of the same type of activity to be basic over several successive time periods. It therefore seems desirable to introduce a profitable type of activity in as many time periods as possible simultaneously. To do this M.A. Saunders and J.A. Tomlin have explored variants of the reduced-gradient method (a nonlinear programming algorithm already implemented by Murtagh and Saunders in MINOS [14]) on these linear problems to change several nonbasic variables simultaneously, in contrast to the standard simplex method which changes only one nonbasic variable at a time. To ensure that the correct nonbasic variables are used, a "special pricing" technique is employed. When the problem is read in, similar activities in different time periods are identified (from the column or variable names) and linked by a circular list. Thus when an activity is priced out and found to have a favorable gradient, the corresponding vectors in other time periods can be easily found and examined, and, if satisfactory, included as candidates to be changed. It is then possible to make a step which introduces an activity in several successive time periods simultaneously.

Preliminary experiments with the above approach have led to a reduction of 20-30% in Phase II iterations when compared to the standard simplex method applied to the type of economic planning models referred to above. It is clear that many tactical variations of the scheme need to be studied.

If it is advantageous to bring in an activity simultaneously in many time periods, then, conversely, it should be advantageous to be able to also force an unprofitable activity to its lower bound in several time periods simultaneously. This is rather more difficult, since one cannot tell whether a whole group of variables can reach their bounds while maintaining a feasible solution (at least, not without incurring a heavy computational cost). The approach we have implemented identifies groups of unprofitable nonbasic activities close to their bounds and forms a direction vector scaled in such a way that if one of these variables reaches its bound then all of them do. This method has had some success in reducing the iteration count when combined with the "special pricing" described above. Again many variations are possible, and very considerable further experimentation is required to refine the methodology and expand on the promising results achieved so far.

While we have concentrated on means of improving the solution path (iteration count) above, another means of improving solution techniques for staircase models is to speed up each step of the simplex algorithm by taking advantage of the special structure of the basis for such problems. As early as 1954, G.B. Dantzig [6] pointed out that such problems exhibit an "almost" square block-triangular basis structure which could be decomposed into a product of a true square block-triangular matrix and another matrix with only a few columns differing from the unit matrix. The advantage of this procedure is that square block-triangular matrices can themselves be very efficiently decomposed to give a very sparse factorization of the basis. A version of this method, employing modern factorization techniques, has been implemented at SOL by A.F. Perold (a graduate student) and is now being tested on problems of significant size. Early indications are that this method of handling the basis can indeed be more efficient than a direct treatment which does not take the staircase structure of models into account. Perold's code is based on SOL's LPM1 linear programming code, as was the nested decomposition code by J.K. Ho and A.S. Manne [12] for the same class of problems. This should facilitate comparison between the specialized simplex and decomposition approaches to these models.

An alternative to all of the above numerical treatments of discrete multi-time-period models is to attempt to solve the underlying continuous time problem, which can be thought of as a linear control problem with state-space constraints. G.B. Dantzig and R.E. Davis are investigating a "continuous simplex method" for such problems. Although progress has been made, much work remains to be done.

Selected Bibliography

[1] Abadie, J.M. (1962), "Dual Decomposition Method for Linear Programs," Comp. Center Case Institute of Technology, July 1962.

[2] Avriel, Mordecai (1976), "Modeling Personal Consumption of Goods in the PILOT Energy Model," Technical Report SOL 76-17, Department of Operations Research, Stanford University, Stanford, California.

[3] Beale, E.M.L. (1963), "The Simplex Method Using Pseudo-Basic Variables for Structured Linear Programming Problems," from Recent Advances in Math. Prog., R. Graves and P. Wolfe, eds., McGraw-Hill.

[4] Bell, E.J. (1964), "Primal-Dual Decomposition Programming," Industrial Engineering Department, University of California, Berkeley, unpublished Ph.D. Thesis.

[5] Bennett, J.M. (1966), "An Approach to Some Structured Linear Programming Problems," Operations Research, Vol. 14, No. 4, July-August, 1966, pp. 636-645.

[6] Dantzig, George B. (1954), "Upper Bounds, Secondary Constraints, and Block Triangularity in Linear Programming (Notes on Linear Programming: Part VIII, X)," The RAND Corporation, RM-1367, October 1954; also in Econometrica, Vol. 23, April 1955, pp. 174-183.

[7] Dantzig, George B. and S.C. Parikh (1975), "On a PILOT Linear Programming Model for Assessing Physical Impact on the Economy of a Changing Energy Picture," Energy: Mathematics and Models, Fred S. Roberts, ed., Proc. SIMS Conference on Energy, Alta, Utah, July 1975, pp. 1-23.

[8] Dantzig, George B. and P. Wolfe (1961), "The Decomposition Algorithm for Linear Programming," Econometrica, Vol. 29, No. 4, October 1961.

[9] Gass, Saul I. (1966), "The Dualplex Method for Large-Scale Linear Programs," Operations Research Center, 1966-15, University of California, Berkeley, June 1966, Ph.D. Thesis.

[10] Geoffrion, A. (1970), "Elements of Large-Scale Mathematical Programming, Part I: Concepts," Management Science, Vol. 16, No. 11, pp. 652-675.

[11] Geoffrion, A.M. (1971), "Large-Scale Linear and Nonlinear Programming," in *Optimization Methods for Large-Scale Systems*, D.A. Wismer, ed., McGraw-Hill, pp. 47-74.

[12] Ho, J.K. and A.S. Manne (1974), "Nested Decomposition for Dynamic Models," Mathematical Programming, Vol. 6, pp. 121-140, 1974.

[13] Lasdon, L.S. (1970), *Optimization Theory for Large Systems*, MacMillan.

[14] Murtagh, B.A. and M.A. Saunders (1976), "Nonlinear Programming for Large, Sparse Systems," Technical Report SOL 76-15, Department of Operations Research, Stanford University, Stanford, California.

[15] Orchard-Hays, W. (1973), "Practical Problems in L.P. Decomposition," in *Decomposition of Large-Scale Problems*, North Holland.

[16] Parikh, Shailendra C. (1976), "Analyzing U.S. Energy Options Using the PILOT Energy Model," Technical Report SOL 76-27, Department of Operations Research, Stanford University, Stanford, California.

[17] Rosen, J.B. (1963), "Primal Partition Programming for Block Diagonal Matrices," Computer Science Division, School of Humanities and Sciences, Stanford University, Technical Report No. 32, November 1963; *Numerishe Math.*, Vol. 6, No. 3 (1964), 250-264.

THE CHALLENGE OF ANALYTIC USE OF COMPUTERS FOR GLOBAL PROBLEMS

William Orchard-Hays
International Institute for Applied Systems Analysis

This year the United States is two centuries old, as you may be tired of hearing by now. Even if one goes back to the early English settlements, this land is only about three and one-half centuries old. That is not very long, really. In terms of generations, about 12 to 14. The small city near Vienna where I have been living for nearly two years goes back to the tenth or eleventh century. There are many active monasteries in Austria and some have been in continuous operation since the eighth or ninth century. So we are a young nation in some ways; yet our political institutions are among the oldest in their present form, and in many areas of technology and business we have led the rest of the world.

The impact of the U.S. on the world has been tremendous. One must live for awhile outside North America to begin really to understand this. Other peoples have watched us--not necessarily their governments--much longer than we have watched them, and still do to a greater degree. We began to be a world power probably about the time of the Spanish-American war, just over three-quarters of a century ago. After World War I, U.S. business became a dominant force in many areas and grew with the world. And it has grown. I was born just before the end of World War I and the population of the world was then much less than two billion, or less than 45% of what it is today. Approximately the same ratio holds for the U.S. and the effect is noticeable.

During and after World War II, of course, the U.S. became the mightiest nation on earth--in almost any way you want to measure except sheer population figures. Yet it was five years after that before computing as we know it today could be said to be in its infancy. (Vuegraph 1.) There were, of course, antecedents going back much further, but these are mainly of historical interest. I started in the field at the RAND Corporation in January, 1951. It was three years later before we had a real computer and we were among the first. I had lived almost half a lifetime before getting into computing and still my career has spanned virtually its entire history. So our field is very young indeed.

Nevertheless, the accomplishments of the computing field during its first quarter century are extremely impressive. Furthermore, with apologies to our many foreign friends and professional associates, computing is almost exclusively an American development. There are three technologies in which we are still clearly supreme: telephone systems, commercial aircraft, and computing. (The list is no doubt longer but, I fear, getting shorter as time goes on.) One flies almost everywhere in American aircraft, and foreign computer manufacturers have all but given up--with the possible exception of Japan and the new imitative line which the Eastern bloc is attempting to create. Some very good software has come out of France, particularly in the MP field, and a little from England, but almost always in connection with American manufacturers or other multi-national corporations. The bulk of system software comes from the U.S. and a large part of application software. (Vuegraph 2, with extemporaneous discussion of evolution of computing technology.)

I think no one in the 1950s really understood the amount of effort required to develop systems, application packages, efficient compilers, and so on. Almost nothing worked right during the whole frustrating decade of the '60s. In retrospect, it is not surprising and we could all be forgiven, I think, for a little self-praise that so much was actually accomplished. But those frustrations often left a bad taste, a deep skepticism, a rigidity of method and, in some cases, bad feelings. One can still see the effects of this.

But the '60s are behind us, and there is no point in dwelling on the mistakes, failures and unrealistic estimates. The solid accomplishments far outweigh them. In fact, the field has progressed so rapidly that it is now possible for one or two people to accomplish in a couple of weeks what would have been considered, only ten years ago, a sizeable project. I

23

have completed several such tasks in the past several months, and surprised even myself. But this doesn't happen always and everywhere. It is surprising how much computing is still done in old-fashioned ways (if we can use that term for so young a field) and how slowly newer techniques spread. At one time, I fought operating systems--not because I didn't understand the problems but, on the contrary, I understood them very well and didn't trust anyone else to solve them in a way satisfactory as a base for my work. Against the advice of knowledgeable colleagues, I long ignored telecommunications, remote processing and interactive systems--except for a couple of especially suitable uses--for the same reason. But in all cases, enough people worked long enough to make the various systems operate in a satisfactory and even elegant and fruitful manner.

Of course, most of us are not interested in the whole computing field. In fact, it is now so extensive that it is almost impossible to comprehend it. However, the math programming field is very rich in itself and its growth has both paralleled and been a component part of the whole experience of the computing field. It had its identifiable beginnings at the same time as computers, that is, in about 1947-8. That was when Dantzig developed the simplex method in connection with planning problems for the U.S. Air Force. As with computers, about five years passed during early developments before linear programming began to blossom into a practical technique. A quarter of a century has passed since then and, for most of this time, LP has pushed the capacity of current computers to the limit. LP and the simplex method are still the foundation stones of the whole field of math programming and the vanguard of OR techniques.

I first started working with George Dantzig in December, 1952. I got the idea somehow that I was supposed to build an automated simplex method and that the principle problems were the amount of arithmetic, maintaining sufficient precision, and the large storage required. We started out modestly. I tried for 25 rows on the old Card-Programmed-Calculator and upped this to 40 when we worked out the product form of inverse. A year later we were trying to do 100 rows on the IBM 701 and, when I solved Alan Manne's 101-row gasoline blending problem, I thought we had really accomplished something. We went to 256 rows on the 704, later 512 and then 1024 rows on the IBM 7090. It was ten years after I started before we had a reasonably automatic system for that size problem, and then not always. Furthermore, the data processing and service routines outweighed the algorithms. I didn't know what I was getting into in 1952.

But then I suspect neither did George Dantzig. The development of mathematical programming in breadth, depth, prestige and applications during the 1960s was fantastic. We had our share of frustrations--decomposition, in spite of much initial interest, extensive software efforts, and a later revival, has still not been perfected nor applied on a large scale; matrix generation techniques, with perhaps even more intensive efforts, still have no widely accepted theoretical or practical basis. This area is now commanding much of my own attention. However, we have an international Mathematical Programming Society with its own Journal and which draws a bigger attendance at its symposia than the whole computing community in the mid-50s. Professor Dantzig's work has been recognized by a Presidential award. All this and much more is well known to you.

Perhaps not so well known is the institute where I am presently working. It is called the International Institute for Applied Systems Analysis or IIASA. This represents a further extension of the field of operations research to an international setting. It is not clear what we should call this whole field of which we are a part. It is not easy to define systems analysis concisely, in fact one program at IIASA is to produce a set of definitive publications for the field. Perhaps of more immediate interest is the genesis and make-up of IIASA. It has been in formal existence only since late 1972 but this was preceded by several years of planning, negotiations and agreements. Professor Howard Raiffa from Harvard was its first director. In his speech at the Council meeting in November 1975, he said the following: "I believed in the concept of IIASA when I was first introduced to the idea by McGeorge Bundy back in 1968. ...Scientific detente then as now is critically necessary if we are to have a sane world." To what extent political detente was a driving force in the formation of IIASA is not clear to me--my first introduction to the Institute was in October 1973 when Dantzig asked me to attend one of its organizational conferences. In any event, it truly is international in character with particular emphasis on shared sponsorship and direction by both East and West. The U.S. and the U.S.S.R. each contribute equally and in by far the largest amounts. There are 13 or 14 other countries who each contribute an equal but much smaller amount. This funding is not done directly as government grants but through Academies of Science or equivalent organizations in the various countries. Austria contributes in a special way through providing facilities at very low or even token rates, which is in conformity with their policy of becoming an international meeting ground. The Institute is housed in the refurbished summer palace--or Schloss, as it is called--of Maria Theresa, at

Laxenburg about 10 miles south of Vienna. (It is quite an experience working in the midst of imperial splendor.) As to its mission, I can do no better than to again quote Raiffa: "I believe IIASA has a mission. It must keep its doors open, so that scientists who have a broad vision of tomorrow's world can communicate with each other. We cannot afford to let these doors shut tightly because of some ephemeral, trivial problems. We must learn today how we can jointly grapple with the stream of equally devastating problems that will surely shake our societies in the not too distant future."

Raiffa's remarkable performance in getting the Institute under way has become almost legendary. He was replaced a year ago by Dr. Roger Levien from the RAND Corporation. The chairman remains Professor Jermen Gvishiani from the USSR Academy of Sciences.

For the past few months, I have been working with the Energy Research Program, headed by Professor Wolf Häfele from the Federal Republic of Germany who is also deputy director of IIASA. Häfele was one of West Germany's leading experts in nuclear energy and is well known in the energy field. The Energy Program is currently the largest at IIASA with about two more years to go under present plans. But there is not time for more details about IIASA's organization and operations.

There are many aspects to the Energy Program but one is the formulation, implementation, coordination and operation of a set of models with emphasis on strategies for a transition from fossil to nuclear fuels. At least that is the way it started out. Häfele and Alan Manne, with some assistance also from Dantzig, produced an energy supply model in 1974. It was a dynamic LP model and there have been several variants and subsidiary models. It remains the centerpiece of the modelling system but many other areas have had to be investigated and models are under development for some of them. These include energy resources, energy demands, long-range economic forecasts (the last two being among the most difficult) and secondary investment requirements. Häfele asked me to coordinate--he calls it orchestrate--the computerization of the entire modelling effort. In a practical sense, this turns out to involve somewhat more than merely computer aspects. When one is trying to coordinate computations with a set of models variously formulated by Russians, Germans, Americans, Austrians, Frenchmen and others, one must become involved with concepts, definitions, nomenclature, feasibility of approach and so on. One must also push aside with some diplomacy certain formulations and levels of detail which are unsuitable for the overall task.

Many of the models are LP models. The Russians, in particular, use LP to an almost unbelievable extent in their national planning and so it is understandable that they take the same approach to international models. We also hope to have not only many component models but versions for at least a few regions of the world with a global model of some kind on top. Häfele has misgivings about the appropriateness of LP for some of these purposes, and so do I, but there is clearly much LP work to do in any event.

It is clear that we will have to solve some models many times on an independent basis in order to get started. It is my hope that we may learn enough this way to be able to circumscribe the meaningful ranges on parameters and devise scenario parts which can be combined for various cases. We don't yet know how to handle the entire system of models in one piece so why not cut it up and solve some of the pieces a number of times to get boundary conditions for the others. Maybe you call this suboptimization but it is better than no solution. However, for this to be meaningful, as many of the models as possible must be put into a consistent framework with respect to style, structure, nomenclature, indexing conventions and so on.

However, my main difficulty right now is that I don't have a suitable computer even for the pieces. I have the software, and there is much other useful software available to us, but not the computers. Considering the nature of the place where I'm working, this shouldn't be so. I am now more than a little envious of all the hardware I've seen in corporate headquarters, investment houses--yes, and universities--which people are really just playing around with. I don't want to take it away from them because there are beneficial side effects--a point I'll return to at the end. But there is something badly wrong with a world economic order that gives its best tools to those who really don't have an essential need and denies them where they are needed for studies with global implications.

At the Math Programming Symposium in Budapest last August, I listened to a Hungarian give a paper on a state planning project he had been involved in. They worked over a year and went through all kinds of devious methods to solve their model. Essentially, it was a big GUB problem. All he needed was a few hours with MPS-III on a 360/65 or better and the problem would have been solved. I spent some hard years developing GUB in MPS-III. Here was someone who really needed it but couldn't get at it for political and economic reasons. It strikes me that there is also something wrong with developing tools and then not making them available when they are needed. The money is spent anyway. If the Hungarians do a better job of planning with our tools, how can this possibly hurt us? Oh, I'm fully aware of the differences between East and West and the difficulties of normal economic relationships. Somehow this doesn't seem very important anymore.

Perhaps you think I'm either off-base or just naive. I think not. As short as the history of our field is, it will be a shorter time still before enormous human calamities begin to occur. I don't mean big wars--God forbid the world becomes embroiled in any more of those, although I could point out a couple of disadvantages we now face if it did happen. But calamities are already happening. Tens or hundreds of thousands of people have already starved in Bangladesh and middle Africa, in just one season. Some people now think India is a hopeless case. Closer to home, the United Kingdom is a catastrophe. Within living memory, the British Empire was the greatest power on earth. It appears that England may soon be reduced to a poor country. Even some Frenchmen feel bad about it. Here at home, we have had one oil crisis--you took it seriously if you were living in Boston at the time. There will be more. It appears likely that OPEC oil prices will go to $15.00 per barrel this month. That is not yet a catastrophe for the U.S. but prices will go higher and higher unless some drastic changes are made in our economic policies--that is, Project Independence and much more must become a reality.

Eduard Teller visited IIASA last month and gave a most wonderful talk on alternate sources of energy. He said he didn't know whether the really serious problems of the world could be solved but he was sure of one thing--they couldn't be solved without plenty of energy available. Furthermore, much of it has to be in a form which is usable in small doses in many places. For example, it would do no good to put a thousand megawatt reactor in an undeveloped country. They simply don't have the wires to carry the electricity away or to distribute it to all the places it is needed. The only form we presently have which is usable this way is petroleum. But if the U.S. keeps gobbling up 45% of the world's production, the price will keep going up putting it out of reach of many nations who can barely pay for what they use now. Furthermore, we only hasten the day and postpone the preparation for it when we will begin to run short ourselves. And unless we drastically change our style of living, that will be a catastrophe here.

I bring these things to your attention to indicate the kind of problems the world must be about solving--and immediately. I will add to Teller's remark and say the really serious problems can't be solved without effective use of computers on a wide scale. Not for keeping accounts, or printing electric bills and bank statements, or even for simulating the aerodynamic qualities of a new airfoil, but for systems analysis or whatever you wish to call it, seriously and expertly applied. That may not be sufficient but it is necessary. Furthermore, the studies, planning and decisions must be international in scope and that means across East-West boundaries, among others. Otherwise, either they will be ineffective or there will be counterplays which come to the same thing. And the world has run out of time for playing such games.

I was talking to a senior French analyst the day after Teller's talk and we got to philosophizing a bit. I said I thought there would be such dislocations in the world during the next quarter century that no one could predict how the global economic order would change. He agreed and even gave some examples from history. I then added that I thought a catastrophe in one area would be bound to have an effect on the rest of the world. Here he shook his head. No, he said, if a million people died in India or Africa, some people in Paris would send a little money to the Post Office for a relief fund but otherwise life would go on just the same. No one would really care. Did the French care when Spain was living in poverty? No, it was a fine place for cheap vacations.

Still, I must believe that the cumulative effect of a series of calamities will be widespread, the more so as they begin to be chronic rather than exceptional. The decline of British power, for example, is probably a tragedy of greater consequence than starvation in some undeveloped country. Why? Because, whether the British were liked or not, they nearly always left things in a better state than they found them. If they can not now manage their own affairs, the world has lost one force for improvement and gained still another problem. At the least, it should be a sobering lesson to us.

I think all of us, even including knowledgeable scientists and high officials of state, have a tendency to disbelieve what seems monstrous and new. We think of such things abstractly unless and until they come down to affecting our daily lives. We might do well to remember Marie Antoinette and the Romanovs. Optimism and courage are great virtues but they lie very close to foolhardiness. It is commendable to be astute and shrewd in business but this should be guided by vision.

The basic outline of approaching world problems was recognized twenty years ago. The late J.D. Williams gave some very easily understood, dramatized papers, unfortunately probably heard or read by very few people. I attended a series of seminars at the American Management Association in 1961 and one lecture was a partly humorous but highly convincing dramatization of the alarming positive rates of change in a variety of areas. (I regret that I have forgotten the speaker's name; he was from Washington.) He presented this regularly to a variety of business and government gatherings. Perhaps he made it too easy to laugh.

History tells us over and over that no arrangement is permanent. It is foolish to believe that because we are strong we can continue to consume far more than our proportionate share of the world's goods. If it were only a matter of others attaining the degree of affluence we enjoy, then it might be considered reasonable for the technological leader to have the most. This has been pretty much our attitude. Western Europe has in fact gone a long way toward catching up though their per capita consumption of energy, for example, is slightly less than half what ours is. In other areas there is a discrepancy of an order of magnitude and, in some cases, almost two orders. Some of these areas must increase energy consumption just in order to provide food. But the ready supply of easily distributed energy forms is becoming limited. Our own strength and prosperity depend on highly developed uses of these same forms. We need not moralize but only consider being placed in competition with people who must have some of what we want simply to survive. In only another 25 years, the population of the world will have increased another 50%, that is, by more people than were living when I was born. If any substantial number of these people improve their living standard by even 10% to 20%--which is almost nothing--the strain on world capacities will be fantastic.

Let me put it another way. There have been extensive, serious and competent studies of world problems for a number of years. Most of these are based on 1967, occasionally 1970, figures. They talk about 5 year, 10 year or 25 year projections, with a few attempts at long-term forecasts for 50 years or longer. But it is this minute very close to 1977. More than five years have passed since 1970, nearly ten since 1967, and the leaves of the calendar keep flipping. It will soon be 1980 and then 1985. Already world population will be at about 5 billion. When one measures this against the time for building an industrial complex, improving a transportation system, resolving political issues, achieving international accord for even modest projects--well it is clear that many babies born this very day are doomed already. There is no such thing as opportunity to improve their lot or freedom to choose their career.

So what can we do? Over the past two decades, we have developed very powerful tools for analysis and planning. A great deal of hardware and software now works and works well. Modelling techniques, though far from perfect, are well advanced. We are now in a position technically to make real inroads on world problems, to undertake the kind of work which originally motivated the development of large computers, and planning and analytical techniques. But we have not yet learned to coordinate their use consistently on a broad scale, nor to convince the actual decision-makers that they can place considerable reliance on results.

We must find ways to make the use of our marvellous tools more effective. First of all, this means use of interactive systems on both large and small computers, tied together with networks in a consistent manner. All these things exist, piecemeal, and are in daily, reliable use. We have the technology but we lack coordination and singleness of purpose. In the West, this is due to business competition--including the universities and research centers which are nothing more or less than large businesses. In Europe, the discontinuities between traditional nations and peoples add further dichotomies. Between the free market countries and the planned-economy bloc, fundamental policy differences exist. Comparing all these more or less developed countries with underdeveloped countries, one finds that the latter don't even understand the game. They know something of the basic tools but very little about motives, incentives, and other driving forces. Of course, there are individual exceptions.

Using computers with common conventions for analytical work can be a very strong unifying force. No doubt much of the work of this kind has been of small value but if it brings diverse peoples together a little, this is in itself an accomplishment. Moreover, some of the efforts are certainly first rate and fruitful. The more widely they can be understood and appreciated, the more quickly we can make effective use of them on a scale appropriate to the global problems we are all facing. In short, our discipline needs more standardization.

Of course, standards mean different things in different contexts. Some are essentially a legalization of what is already considered good practice, such as state commissions and examining boards over a wide range from barbers to lawyers and doctors. For more technically complicated areas or when massive or tedious data must be gathered, we have such institutions as the National Bureau of Standards and the American Standards Association. But with apologies to our hosts, these approaches do not apply to the present discussion.

De facto standards--usually due to economic forces--are often among the most effective. Several years ago there was much discussion of electronic modes of recording on magnetic tape. Many people claimed IBM's methods were not very good. I don't know whether they were or not but they worked and, to stay in business, other manufacturers had to be compatible with them. Now one can put a tape in his brief-case, travel over a large part of the world and have the tape read by a computer in some remote place. This is a great advantage. I think no one would claim today that the MPS/360 input formats are very good, but for the same reasons, they are an effective standard. As a result, problems can be shipped around and solved on various computers. Furthermore, it only takes one word to describe the format, another great advantage.

Unfortunately, economic forces can also work the other way. One sometimes suspects that economists, operations researchers, and other system analysts--not to speak of software designers--are just as happy if their techniques cannot be measured and compared too clearly and precisely, another result of intense competition. But competition won't solve today's problems; collaboration is much more to the purpose.

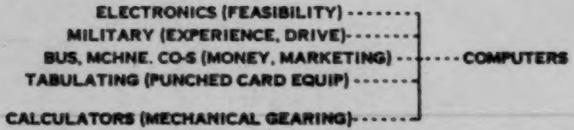
In the task I'm presently engaged in, it is necessary to devise some sort of standards and, one way or another, to enforce their use. To do this, they must be logically consistent and explainable--which is perhaps the best kind of standardization. I have discovered that this is not an easy task but, more importantly, the attempt leads to considerable clarification of fundamental ideas. Let me illustrate this with the problem of identifiers.

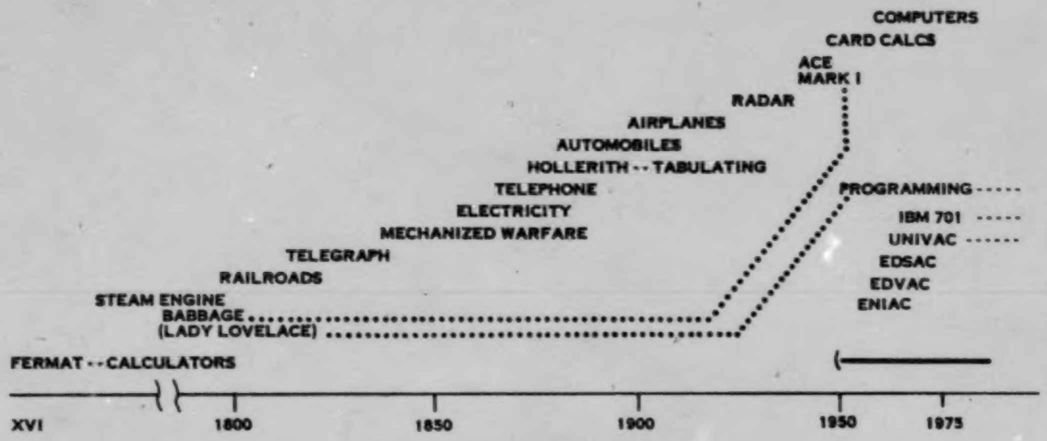
As most or all of you know, an LP model has got to have unique row and column identifiers and, in a practical sense, these are limited to 8 characters with only 36 to 38 graphics available. Since some formulations require five or six indices, one must be extremely frugal with encoding schemes. This forces one to study very carefully just what it is that one is trying to represent and this, in turn, leads to some rather deep considerations. One runs into such problems as what constitute substances, what can be called processes, is there a fundamental distinction between primary and secondary conversions, what are capacities and how are they related to other variables? What is capital and what is labor? These last two gave me considerable trouble but finally I thought I had them correctly classified. I spent most of a Sunday reading the encyclopaedia to see if my conclusions were correct and found that essentially they were. I learned much more which I would never have comprehended if I hadn't been trying to figure out how to automate matrix and report generation and to make them consistent over a range of models. Some economists would do well to go through the same drill to clarify their own thinking.

Once the manipulation of the models is conveniently automated, we can then begin paying attention to the really important questions--the validity of data and its meaning, the effect of variations, i.e. sensitivity analysis, the effect of changes in hypotheses and how this relates to model structure and its projection in reality. It must be possible to sit down at a console and, during a morning, get solutions to several variants of a model, with human interaction with the computer quickly and cryptically communicated. This means interactive systems and, in most situations--certainly ours--it means reliable telecommunication facilities. Only then can we begin to get some feel for the possible behavior of our complicated world.

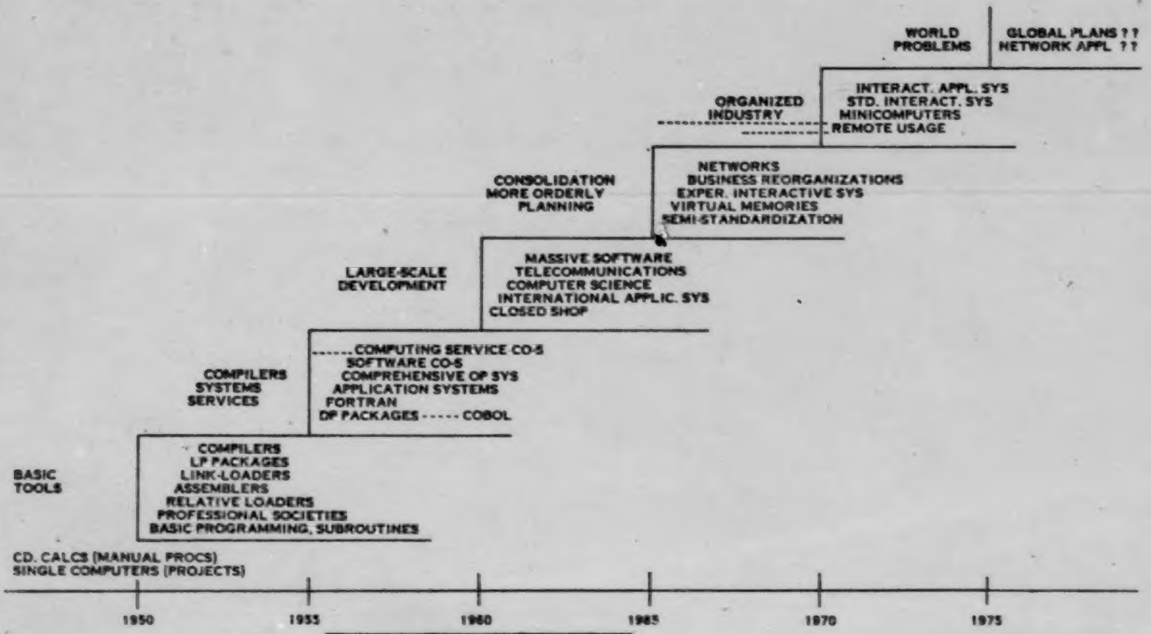
Thus the computer, used as an analytical tool--in fact, almost as a colleague--will be not only a computational engine but a unifying and standardizing force. There will still be plenty of differences of opinions but the issues will be clearer and facts--as nearly as we can approximate them--will stand out.

But all this will happen only if those of us who are skilled in and motivated by the effective use of computers begin to assume leadership. The world is buried in scientific and technical journals and there is no end to ever-multiplying complexity of thought and confusion of detail. It is almost a sickness. The need now is to take hold of all our skills, tools, and proven techniques and mobilize them to the best of our ability in order to clarify issues, influence meaningful decisions, promote rational cooperation, and continually sharpen our perceptions. We have nothing to lose and we might just discover a world order that is workable.

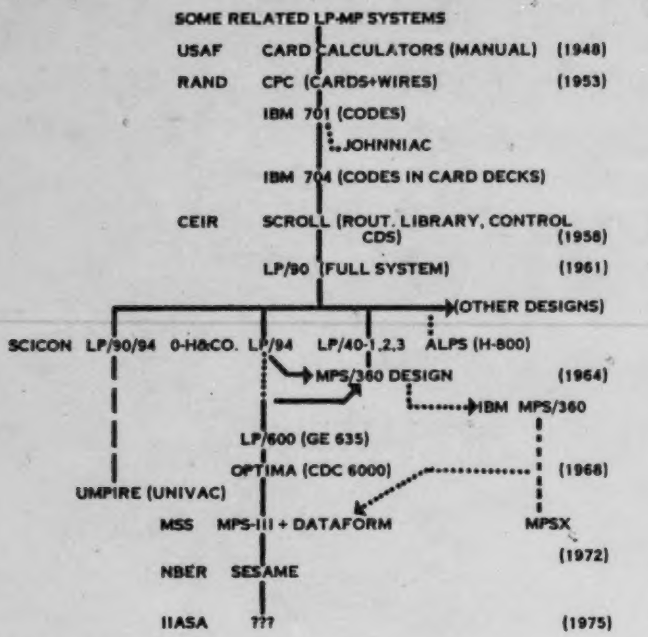

ELECTRONICS (FEASIBILITY)
MILITARY (EXPERIENCE, DRIVE)
BUS. MCHNE. CO-S (MONEY, MARKETING)
TABULATING (PUNCHED CARD EQUIP)
CALCULATORS (MECHANICAL GEARING)
COMPUTERS


COMPUTERS
CARD CALCS
ACE
MARK I
RADAR
AIRPLANES
AUTOMOBILES
HOLLERITH -- TABULATING
TELEPHONE
ELECTRICITY
MECHANIZED WARFARE
TELEGRAPH
RAILROADS
STEAM ENGINE
BABBAGE
(LADY LOVELACE)
FERMAT -- CALCULATORS
PROGRAMMING
IBM 701
UNIVAC
EDSAC
EDVAC
ENIAC
XVI
1800
1850
1900
1950
1975

1. DEVELOPMENT OF COMPUTERS

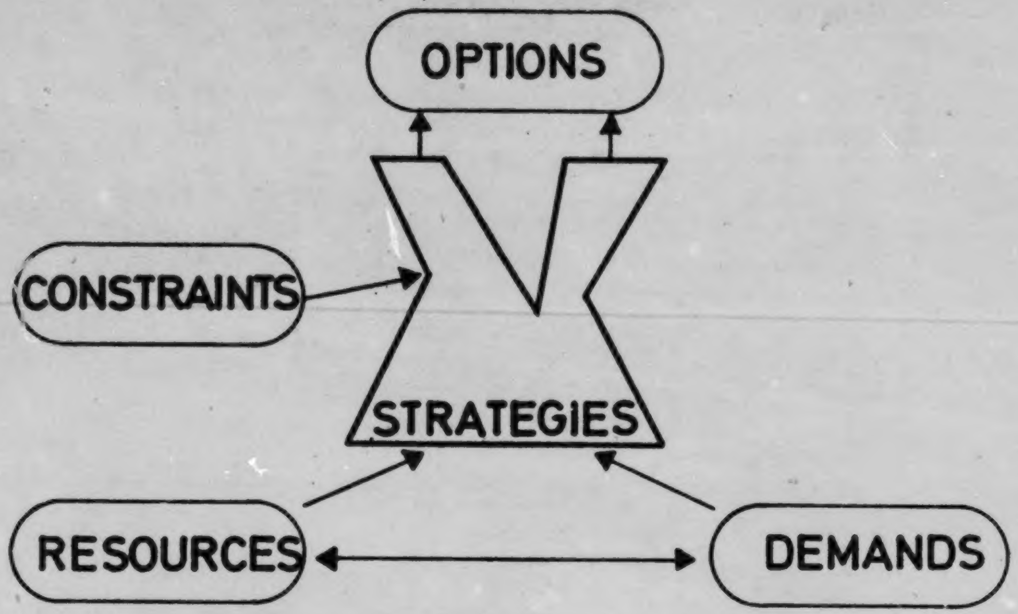

WORLD PROBLEMS
GLOBAL PLANS ??
NETWORK APPL. ??
ORGANIZED INDUSTRY
INTERACT. APPL. SYS
STD. INTERACT. SYS
MINICOMPUTERS
REMOTE USAGE
CONSOLIDATION MORE ORDERLY PLANNING
NETWORKS
BUSINESS REORGANIZATIONS
EXPER. INTERACTIVE SYS
VIRTUAL MEMORIES
SEMI-STANDARDIZATION
LARGE-SCALE DEVELOPMENT
MASSIVE SOFTWARE
TELECOMMUNICATIONS
COMPUTER SCIENCE
INTERNATIONAL APPLIC. SYS
CLOSED SHOP
COMPILERS SYSTEMS SERVICES
COMPUTING SERVICE CO-S
SOFTWARE CO-S
COMPREHENSIVE OP SYS
APPLICATION SYSTEMS
FORTRAN
DP PACKAGES ----- COBOL
BASIC TOOLS
COMPILERS
LP PACKAGES
LINK-LOADERS
ASSEMBLERS
RELATIVE LOADERS
PROFESSIONAL SOCIETIES
BASIC PROGRAMMING, SUBROUTINES
CD. CALCS (MANUAL PROCS)
SINGLE COMPUTERS (PROJECTS)
1950
1955
1960
1965
1970
1975

2. DEVELOPMENT OF PROGRAMMING

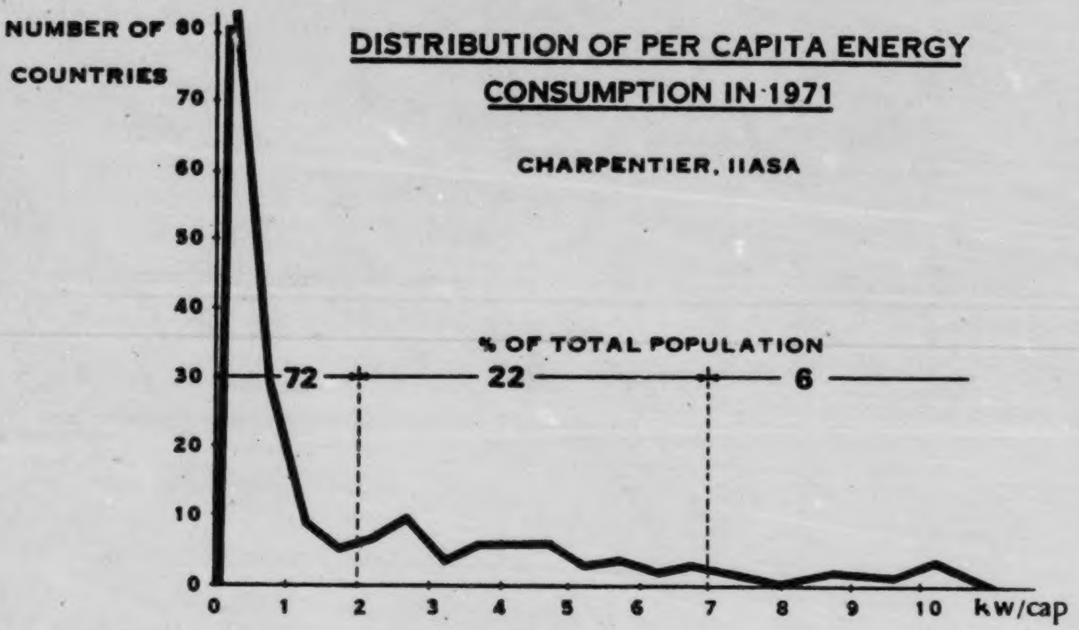

SOME RELATED LP-MP SYSTEMS
USAF CARD CALCULATORS (MANUAL) (1948)
RAND CPC (CARDS+WIRES) (1953)
IBM 701 (CODES)
JOHNNIAC
IBM 704 (CODES IN CARD DECKS)
CEIR SCROLL (ROUT. LIBRARY, CONTROL CDS) (1958)
LP/90 (FULL SYSTEM) (1961)
(OTHER DESIGNS)
SCICON LP/90/94
0-H&CO. LP/94
LP/40-1,2,3
ALPS (H-800)
MPS/360 DESIGN (1964)
IBM MPS/360
LP/600 (GE 635)
OPTIMA (CDC 6000)
(1968)
UMPIRE (UNIVAC)
MSS MPS-III + DATAFORM
MPSX
(1972)
NBER SESAME
IIASA ???
(1975)

3. EXAMPLE OF EVOLUTION OF APPLICATION SYSTEMS

	RESOURCES & ENVIRONMENT	HUMAN SETTLEMENTS & SERVICES	MANAGEMENT & TECHNOLOGY	SYSTEM & DECISION SCIENCES
GLOBAL ENERGY SYSTEMS				
UNIVERSAL REGIONAL DEVELOPMENT				


OPTIONS
CONSTRAINTS
STRATEGIES
RESOURCES
DEMANDS

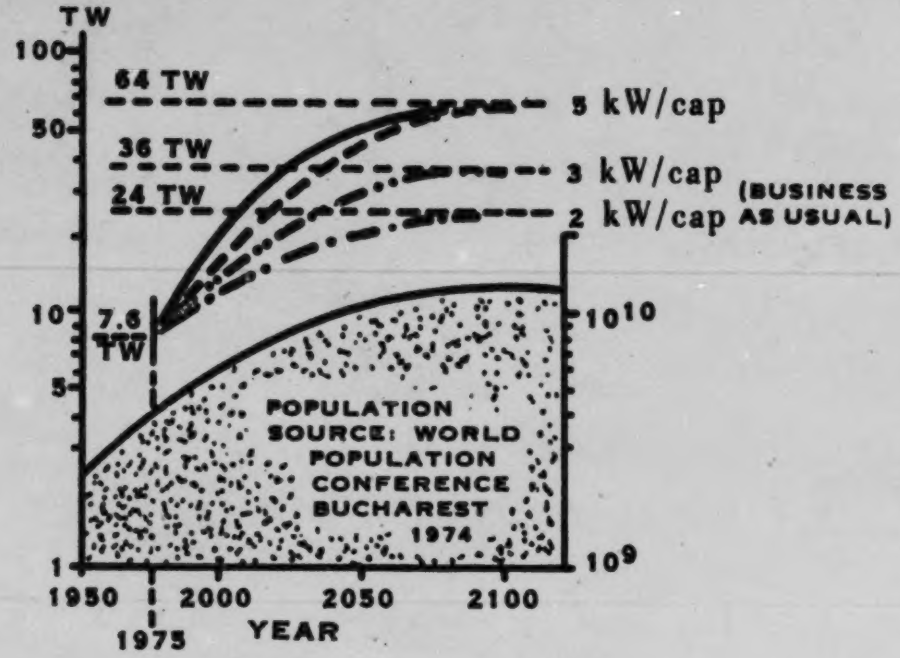
THE APPROACH TO ENERGY SYSTEMS

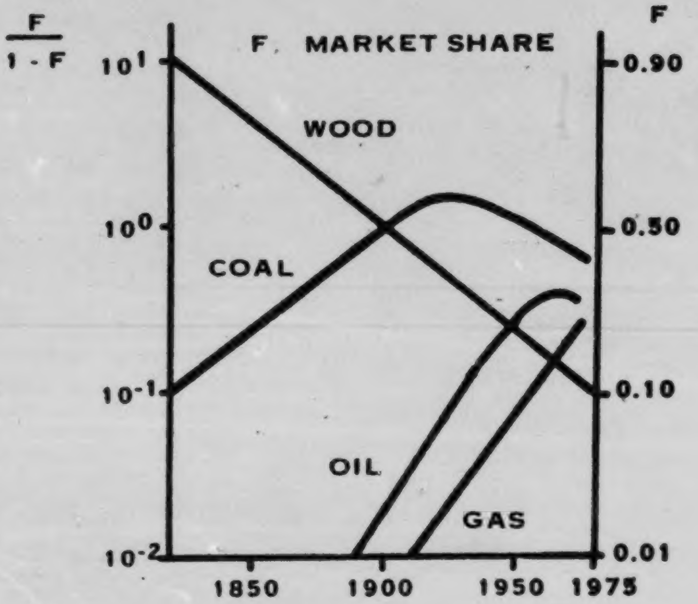


DISTRIBUTION OF PER CAPITA ENERGY CONSUMPTION IN 1971

CHARPENTIER, IIASA

GLOBAL ENERGY SCENARIOS





ENERGY MARKET PENETRATIONS: WORLD

MARCHETTI, IIASA

IMPLEMENTATION AND APPLICATION OF A NESTED DECOMPOSITION ALGORITHM*

James K. Ho

Applied Mathematics Department
Brookhaven National Laboratory
Upton, NY 11973

ABSTRACT

This paper considers a nested decomposition algorithm for multi-stage linear programs with the staircase structure. Computational aspects of the algorithm essential to an efficient implementation are discussed. Experience with using experimental codes of the algorithm on problems arising from real applications, such as energy models, engineering design, and dynamic traffic control is presented. It is observed that nested decomposition can be an efficient technique for large-scale systems.

---

*Work performed under the auspices of the ERDA.

1. The Staircase Algorithm

We consider the linear programming problem of minimizing

$$\sum_{t=1}^{T} c_t x_t$$

$$\text{subject to} \quad A_1 x_1 = d_1$$

$$B_{t-1} x_{t-1} + A_t x_t = d_t, \quad t = 2,\dots, T \qquad (1)$$

$$x_t \geq 0, \quad t = 1,\dots, T$$

where c_t is $1 \times n_t$, x_t is $n_t \times 1$, A_t is $m_t \times n_t$, B_t is $m_{t+1} \times n_t$ and d_t is $m_t \times 1$ in dimensions.

In [3] and [7], the Staircase algorithm is developed from an application of nested decomposition to (1). Using this algorithm, the original problem is replaced by a sequence of smaller, independent subproblems coordinated by primal (proposals) and dual (prices) information in the sense of Dantzig and Wolfe [2].

A cycle of the Staircase algorithm, indexed by k, consists of T subproblems denoted by SP_t^k, $t = 1,\dots, T$, and defined as follows:[1]

$$SP_t^k \left\{ \begin{array}{lr} \text{minimize} \quad (c_t - \pi_{t+1}^k B_t)\, x_t^k + p_t \lambda_t^k = z_t^k & (2) \\ \text{subject to} \quad A_t x_t^k + Q_t \lambda_t^k = d_t & (3) \\ \delta_t \lambda_t^k = 1 & (4) \\ x_t^k, \quad \lambda_t^k \geq 0 & \end{array} \right.$$

for $t = 1,\dots, T$

The various terms in the subproblems are defined recursively as follows, preceded by their dimensions. A prime denotes a transpose.

$$\left. \begin{array}{ll} \text{scalar:} & p_2^k = c_1 \; x_1^k \\ \text{scalar:} & p_t^k = c_{t-1} x_{t-1}^k + \sum_{j=1}^{k-1} p_{t-1}^j \pi_{t-1}^{jk}, \quad (t = 3,\dots, T) \\ 1 \times k: & p_t = (0, p_t^1,\dots, p_t^{k-1}), \quad (t = 2,\dots, T) \end{array} \right\} \quad (5)$$

$$\left. \begin{array}{ll} m_t \times 1: & q_t^k = B_{t-1} x_{t-1}^k \\ m_t \times k: & Q_t = [0, q_t^1, q_t^2,\dots, q_t^{k-1}], \quad (t = 2,\dots, T) \end{array} \right\} \quad (6)$$

---

[1] For SP_1^k, delete the terms involving λ_1^k and also equation (4). For SP_T^k, delete the term $\pi_{T+1}^k B_T$ from (2).

$k \times 1$: $\lambda_t^k = (1, \lambda_t^{1k}, \ldots, \lambda_t^{k-1,k})'$, $(t = 2, \ldots, T)$ (7)

$$\left.\begin{array}{l} \text{scalar: } \delta_t^k = \binom{1}{0} \text{ if } (x_{t-1}^k, \lambda_{t-1}^k) \text{ is a } \left\{\begin{array}{c}\text{feasible}\\ \text{homogeneous}\end{array}\right\} \text{ solution} \\ \qquad (t = 2, \ldots, T) \\ 1 \times k\text{: } \delta_t = (1, \delta_t^1, \delta_t^2, \ldots, \delta_t^{k-1}) \quad , (t = 2, \ldots, T) \end{array}\right\} (8)$$

$1 \times m_t$: π_t^k is the vector of dual variables for (3).

scalar: σ_t^k is the dual variable for (4).

SP_t^k is read as subproblem t at cycle k. SP_t denotes subproblem t at an unspecified cycle. The $(1 + m_t) \times 1$ vector (p_t^k, q_t^k) is called a proposal from SP_{t-1}^k to SP_t^k. The prices for SP_t^k are given by $(\pi_{t+1}^k, \sigma_{t+1}^k)$.

The three phases of the Staircase algorithm are summarized below.

Phase 1:

Step (i): Set t = 1.

Step (ii): Start with an artificial basis for SP_t. Set the objective to be the sum of the infeasibilities in SP_t. Use the Phase 2 procedure to solve the subsystem $[SP_1, \ldots, SP_t]$. If the optimal value of the objective > 0, stop: problem is infeasible. Otherwise, go to Step (iii) if $t < T$; go to Phase 2 if $t = T$.

Step (iii): Generate a proposal for SP_{t+1}. Set $t = t + 1$. Go to Step (ii).

Phase 2:

Step (i): Set k = 1.

Step (ii): Set t = T.

Step (iii): Solve SP_t^k. If $t < T$, send a proposal (if any) to SP_{t+1}^{k+1}. If $t > 1$ and subproblem is bounded, send prices to SP_{t-1}^k, otherwise go to Step (ii); if $t = T$ and subproblem is unbounded, stop: problem is unbounded. If $t = 1$, go to Step (v).

Step (iv): Set $t = t - 1$. Return to Step (iii).

Step (v): Test for optimality: $z_T^k = \sum_{t=1}^{T} \pi_t^k d_t$. If optimal, go to Phase 3. Otherwise, set $k = k + 1$; go to Step (ii).

Phase 3[2]:

Step (i): Set $t = T$, $y_T = x_T^k$, compute $d_T - A_T y_T$.

Step (ii): Set $t = t - 1$. Solve SY_t. If $t = 1$, stop. Otherwise compute $d_t - A_t y_t$. Return to Step (ii).

A flow diagram of the algorithm is given in Figure 1.

2. Implementation

The computational efficiency of an implementation of the Staircase algorithm depends essentially on the following three aspects:

(i) data structure: This prescribes the amount of data required to define a subproblem, and the amount of data transfer required to update a subproblem.

(ii) solving a subproblem: This is how the subproblems are to be solved as linear programs.

(iii) coordinating information: This is how prices are incorporated and how proposals are generated in a subproblem.

---

[2] Define for $t = T - 1, \ldots, 1$,

$$SY_t: \begin{array}{lll} \text{minimize} & c_t y_t + p_t \mu_t & \\ \text{subject to} & A_t y_t + Q_t \mu_t & = d_t \\ & B_t y_t & = d_{t+1} - A_{t+1} y_{t+1} \\ & \delta_t \mu_t & = 1 \\ & y_t, \quad \mu_t & \geq 0 \end{array}$$

For SY_1 delete terms involving μ_t.

For our experimental codes, we adopted the guide-line of making full use of advance simplex techniques in linear programming for solving the subproblems (see e.g. [11], [14]). Efficient designs for (i) and (iii) compatible with such a scheme are then identified. Although we have been limited by the scope of this research to give certain considerations to convenience of programming, the underlying concepts are nonetheless general and should be applicable to even the most sophisticated implementation.

3. Data Structure

Large scale problems are usually very sparse. The density of nonzero entries in the constraint matrix is typically less than one percent [1]. Although the nonzero entries concentrate in blocks for staircase structures, the subproblems defined on these blocks should still be sparse. For example an 0.1% density in a 10-period 1,000 × 10,000 (row by column) problem implies an average density of 0.526% for the non-zero blocks. Therefore, the subproblem data should be stored in packed form. Many schemes are available for storing only the nonzero elements (see e.g. [8], [12]). Since our algorithm consists mainly of column operations, we use a column packing scheme. The nonzeros are packed in a vector (one-dimensional array), by column order. A second vector of the same length contains the row indices of the corresponding entries in the first vector. A third vector gives for each column the position of its first entry in the other two vectors. In the Staircase algorithm, the subproblems are modified by the addition of proposals, which form new columns in the constraint matrices. With the above scheme, appending a new column is done by simple extension of the three vectors.

In general, the data for each subproblem can be classified as follows:

(i) constraint type data,

(ii) right-hand-side data,

(iii) basis data,

(iv) constraint matrix data,

(v) proposal data.

The first two classes remain unchanged from one cycle to the next during Phase 1 and Phase 2. They are modified in Phase 3. The amount of basis data that needs to be stored depends on the subproblem solution procedure. In our case, these are simply index vectors identifying a basis. Class (iv) consists of data for the current constraint matrix, while (v) contains data for the new columns to be added to form a new subproblem in the following cycle. The necessity to differentiate between (iv) and (v) depends on the relative efficiency in concatenating data records, which is in turn determined by the data storage device and mode of data transfer being used.

Subproblem data are stored out of core. Regions in out-of-core memory are designated for the various classes of data. Each region is subdivided into T subregions, corresponding to the number of periods in the problem. To solve a subproblem, data from the appropriate subregions are read into the work region (scratch space) in core. Outputs from the subproblem are written into the appropriate subregions as data for later cycles. The data flow for a subproblem is illustrated in Figure 2.

4. Solving a Subproblem

To solve the subproblems efficiently as linear programs, we use a product-form-inverse (PFI) revised simplex routine with an inversion subroutine designed to produce sparse representations of the basis inverse. This routine is based on LPM1, an in-core LP code written by J. Tomlin in 1970 [14].

5. Coordinating Information

The typical subproblem can be arranged in the form shown in Figure 3 where ~ denotes a non-binding row. No pivot operation is to be performed on such a row. Apart from being consistent with the data structure described in Section 3, this full subproblem formulation has the advantage of allowing (a) dual prices to be incorporated into the objective function without any change in original data and (b) primal proposals to be inferred from the updated right-hand-sides of the non-binding rows. To show this, we use the simplifying notation in Table 1.

Now, suppose we maintain a basis for the full subproblem including the non-binding rows. Such a basis will have the following form:

$$\hat{M} = \begin{bmatrix} 1 & \hat{c} & 0 \\ 0 & \hat{A} & 0 \\ 0 & \hat{B} & I \end{bmatrix} \qquad (9)$$

where the entry 1 and the identity submatrix I indicate that the slacks of the non-binding constraints are always basic since no pivoting is done on such rows. The basis inverse is then

$$\hat{M}^{-1} = \begin{bmatrix} 1 & -\hat{c}\hat{A}^{-1} & 0 \\ 0 & \hat{A}^{-1} & 0 \\ 0 & -\hat{B}\hat{A}^{-1} & I \end{bmatrix} \qquad (10)$$

The objective function for SP_t^k, as given by (2) is (dropping the superscript k for convenience),

$$z_t = (c_t - \pi_{t+1} B_t)x_t + P_t\lambda_t \ ,$$

which is to be minimized. In the present notation,

$$z_t = (c - \pi_{t+1}B)x \ . \qquad (11)$$

We argue that it is not desirable to set up z_t explicitly. First, a direct multiplication of π_{t+1} and B would make it necessary to distinguish between data in A and B. This is cumbersome since the matrix data are packed column by column. Secondly, to update the objective, we need to store the original c separately since this is used every time we compute a new z_t. Finally, updating requires the accommodation of new nonzeros. This cannot be done efficiently in the data structure being used. All these complications can be circumvented as follows. Recall that without the $\pi_{t+1}B$ term in the objective, the simplex multiplier π for the basis $\hat{M}$ would satisfy

$$\pi\hat{M} = \overbrace{(1,\ 0, \ldots,\ 0)}^{m \text{ components}}$$

This gives a reduced cost of 1 to s_0, the slack variable in the objective row which is always basic since the objective row is nonbinding, and zero reduced costs to the other basic variables.

Suppose we now require π to satisfy

$$\pi\hat{M} = \underbrace{(1,\ 0,\ \underbrace{-\pi_{t+1}}_{m_{t+1}})}_{m} \quad \text{components} \qquad (12)$$

so that[3]

$$\pi = (1,\ 0,\ -\pi_{t+1})\hat{M}^{-1}$$

$$= (1,\ 0,\ -\pi_{t+1}) \begin{bmatrix} 1 & -\hat{c}\hat{A}^{-1} & 0 \\ 0 & \hat{A}^{-1} & 0 \\ 0 & -\hat{B}\hat{A}^{-1} & I \end{bmatrix}$$

$$= (1, -(\hat{c} - \pi_{t+1}\hat{B})\hat{A}^{-1},\ -\pi_{t+1}) \ .$$

Letting $\pi_t = (\hat{c} - \pi_{t+1}\hat{B})\hat{A}^{-1}$, we have $\pi = (1, -\pi_t, -\pi_{t+1})$ and the reduced costs are

$$\bar{c} = \pi M = c - (\hat{c} - \pi_{t+1}\hat{B})\hat{A}^{-1}A - \pi_{t+1}B$$

$$= (c - \pi_{t+1}B) - \pi_t A \qquad (14)$$

so that we are effectively using z_t.

Note that (12) implies that the reduced costs of the slacks variables in the last m_{t+1} rows are precisely $-\pi_{t+1}$. These variables are always basic since the corresponding rows are nonbinding. Therefore, we can interpret (12) as the setting of prices on the inputs and outputs described by the matrix B.

Next, consider the updated right-hand-side given by (15) where $\hat{x}$ gives the values of the basic variables in the current extreme point solution x of SP_t^k. The nonbasic variables in x are, of course, at zero value.

---

[3] This is the backward transformation or BTRAN step in the revised simplex method.

$$\beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = \hat{M}^{-1} d = \begin{bmatrix} -\hat{c}\hat{A}^{-1} d_t \\ \hat{A}^{-1} d_t \\ -\hat{B}\hat{A}^{-1} d_t \end{bmatrix} = \begin{bmatrix} -\hat{c}\hat{x} \\ \hat{x} \\ -\hat{B}\hat{x} \end{bmatrix} \tag{15}$$

Therefore, according to (5), (6), and (15), a proposal corresponding to x is given by

$$\begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} cx \\ Bx \end{bmatrix} - \begin{bmatrix} \hat{c}\hat{x} \\ \hat{B}\hat{x} \end{bmatrix} = - \begin{bmatrix} \beta_1 \\ \beta_3 \end{bmatrix} \tag{16}$$

which, apart from the sign, is part of the updated right-hand-side β.

Proposals corresponding to extreme ray solutions of SP_t^k can be generated as follows. Suppose a column in M, say

$$N = \begin{bmatrix} c^s \\ A^s \\ B^s \end{bmatrix}$$

is priced out to be the pivot column. Its transformation[4] with respect to the basis $\hat{M}$ is

$$\bar{N} = \begin{bmatrix} \bar{N}_1 \\ \bar{N}_2 \\ \bar{N}_3 \end{bmatrix} = \hat{M}^{-1} N = \begin{bmatrix} 1 & -\hat{c}\hat{A}^{-1} & 0 \\ 0 & \hat{A}^{-1} & 0 \\ 0 & -\hat{B}\hat{A}^{-1} & I \end{bmatrix} \begin{bmatrix} c^s \\ A^s \\ B^s \end{bmatrix}$$

$$= \begin{bmatrix} c^s - \hat{c}\hat{A}^{-1} A^s \\ \hat{A}^{-1} A^s \\ B^s - \hat{B}\hat{A}^{-1} A^s \end{bmatrix} \tag{17}$$

If $\bar{A}^s = \hat{A}^{-1} A^s \leq 0$, then SP_t^k is unbounded from below, with

$$x_h = (0, \ldots, 0, \underset{s^{th}\ \text{position}}{1}, 0, \ldots, 0, \underset{\substack{j^{th}\ \text{position} \\ \text{where } x_j \text{ is} \\ \text{basic in } i^{th} \\ \text{row}}}{(-\bar{A}^s)_i}, 0, \ldots, 0) \tag{18}$$

as a homogeneous solution on the extreme ray (cf. Theorem 2.1, p. 35 in [13]). By definition, the extreme ray is simply $x_h/(e \cdot x_h)$ where $e = (1, \ldots, 1)$. Observe that by (5), (6), (17) and (18), the proposal corresponding to x_h is

$$\begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} cx_h \\ Bx_h \end{bmatrix} = \begin{bmatrix} c^s - \hat{c}\bar{A}^s \\ B^s - \hat{B}\bar{A}^s \end{bmatrix} = \begin{bmatrix} c^s - \hat{c}\hat{A}^{-1} A^s \\ B^s - \hat{B}\hat{A}^{-1} A^s \end{bmatrix} = \begin{bmatrix} \bar{N}_1 \\ \bar{N}_3 \end{bmatrix} \tag{19}$$

which is part of the transformed column $\bar{N}$. The proposal corresponding to the extreme ray can, of course, be obtained from (19) by scaling with $1/(e \cdot x_h)$. However, this is not necessary and we may simply send the proposal in (19) to SP_{t+1}^{k+1}. This is equivalent to an implicit scaling of the corresponding λ variable in SP_{t+1}^{k+1} by $e \cdot x_h$, hence there is no loss of generality.

Finally, to determine whether a proposal is profitable to SP_{t+1}^{k+1} according to the prices π_{t+1}, we have to test whether[5]

$$\delta_t = \begin{cases} z_t - \sigma_{t+1} < 0 \\ \text{for extreme point proposals ;} \\ z_t \quad < 0 \\ \text{for extreme ray proposals.} \end{cases}$$

This second case is always satisfied since z_t is unbounded from below. Hence, no computation is required. For the first case we compute

---

[4] This is the forward transformation or FTRAN step in the revised simplex method.

[5] In the Staircase algorithm, "proposal" and "profitable proposal" are used synonymously, so that strictly speaking, the vector defined in (16) is a proposal only if it passes the test.

$$\xi_t = \hat{c}\hat{x} - \pi_{t+1}\hat{B}\hat{x} - \sigma_{t+1}$$

$$= -[\theta_1 - \pi_{t+1}\theta_3] - \sigma_{t+1}\,, \qquad (20)$$

which is essentially the evaluation of an inner product.

We have shown that as the result of applying the revised simplex procedure to the full subproblem

$$\begin{aligned} &\text{minimize} && cx \\ &\text{subject to} && \begin{bmatrix} A \\ B \end{bmatrix} x \overset{=}{=} d \qquad (21) \\ & && x \geq 0 \end{aligned}$$

(a) the dual information π_{t+1} is efficiently utilized as part of the prices π for (21);

(b) the primal information $\begin{bmatrix} P \\ q \end{bmatrix}$ is efficiently generated, being a by-product of right-hand-side updating in the case of extreme points, and forward transformation of a pivot column in the case of extreme rays.

6. Refinements

Within the basic framework summarized in Section 1, many refinements of the Staircase algorithm are possible. Such modifications are called computational strategies. Some are designed to accelerate convergence; others to reduce storage requirements and the amount of data transmission. Most of them are motivated by heuristics and have to be validated empirically. Moreover, good strategies for one class of problems may turn out to be poor ones for another. Therefore, it is important to parametrize such refinements so that a process of fine tuning is possible for any given class of problems. We identify three important strategies here. For more detail, the reader is referred to [3].

(a) Degree of Decomposition:

For a given amount of core storage, it is often feasible to vary the number of subproblems by grouping two or more as a single stage. A higher degree of decomposition gives smaller subproblems which are easier to solve, but is likely to require more interactive adjustments before obtaining an optimal coordination among the subproblems. Initial experience with the Staircase algorithm [3] suggests that whenever nested decomposition algorithms are intended for routine applications on a particular class of problems with a fixed amount of core storage, one should predetermine empirically a good strategy for the degree of decomposition.

(b) Multi-proposal Generation:

By generating (when possible) more than one proposal from each subproblem, convergence may be accelerated. However, if too many proposals are transmitted in a cycle, the inferior ones, which may never become useful, simply cause an unnecessary increase in the size of a subproblem. Moreover, proposals that are too similar may give rise to numerical instability. Therefore, a heuristic procedure is required to select a limited number of proposals. A limit of five provided good results in most cases we encountered.

(c) Proposal Purging:

As proposals are introduced, the growing size of a subproblem may cause difficulties for in-core storage. As far as the optimization of SP_t^k is concerned, the only proposals that need be kept are the currently basic ones (for feasibility) and the latest profitable ones (for improvement). All others could be dropped as they will be generated again if and when they become profitable on later cycles. We use a scheme to purge as many non-basic proposals as necessary to keep the subproblem size within limits determined by core availability. However, a modification of Phase 3 is also required to allow for proposal purging.

7. The Experimental Codes

Two experimental codes, named SC73 and SC74 have been written in FORTRAN. Input data is in standard MPS format plus a section on information characterizing a pattern of decomposition, except for the data handling features. The two codes are identical.

SC73 runs on an IBM 360/91 at the Stanford Linear Acceleration Center. It requires approximately 200K bytes of core storage when dimensioned for problems with a maximum of 20 periods, each having up to 500 rows (as a full subproblem) and 3000 nonzero coefficients (including proposals). Secondary storage is on 220 tracks (one track = 7294 bytes) of IBM type 2314 magnetic disk. Data transmission is by direct access I/O using variables length records with a block size of 7294 bytes.

SC74 runs on the CDC 6000 and 7000 series computers. It requires approximately 35K words of SCM core storage when dimensioned for problems with a maximum of 10 periods, each having up to 500 rows (as a full subproblem) and 6000 nonzero coefficients (including proposals). Secondary storage requires 170K words of ECS (extended core storage) or LCM (large core memory). Block transfer of data between SCM and LCM is used.

For comparison with a direct simplex approach, we used MPS/360 and LPM1 [14], the latter being the simplex procedure used in SC73 and SC74. Roughly the same storage configuration is used in each comparative run.

8. Experience with Applications

This section presents computational experience with the successful application of the Staircase algorithm to three classes of multi-stage linear programs. In each case, we describe the nature of the problem briefly and summarize the performance of the Staircase algorithm as compared to a direct simplex approach.

(a) Optimal Design of Multi-stage Structures [4]:

The problem is to design multi-stage planar trusses for minimal weight over a class of feasible member sizes and configurations. The design is based on limit analysis subject to a single set of loads. The variables x_t represent forces in members of stage t in the truss. The equilibrium conditions for stage t are expressed through A_t while B_t represents the coupling between stage t and t + 1. The external loads are given in d_t. Typically, these problems have a large number of columns in proportion to the number of constraints. Therefore, the efficiency of the Staircase algorithm could be due partly to a partial-pricing effect. See Table 2.

(b) Dynamic energy model [5]:

These test problems are derived from a staircase version of Manne's model of U.S. options for a transition from oil and gas to synthetic fuels [9]. They seek minimum-cost strategies to meet future energy demands under a series of alternative scenarios. The latter depends on estimates of the remaining quantities of domestic oil and gas resources, and the technical and environmental feasibility of new methods for synthetic fuel production. The variables are production capacities and investment in the energy sectors. The single-period constraints exert bounds on new capacity introduction rates and relate production to final demands in energy output. The dynamic constraints relate capacity inventories and investment, model the nuclear cycle, and exert bounds on cumulative resource extraction.

The model has 16 periods representing five-year intervals from 1970 to 2045. However, a four-period decomposition is used for reasons explained in section 6a. All new technologies are allowed in problem 4A (see Table 3) while most of them are suppressed in problem 2A. The performance of the direct simplex approach (LPM1) reflects this variation in complexity. Whereas, by decomposition, such effects are "felt" by each subproblem from the start. This may explain why SC74 took roughly the same amount of time for all four problems.

(c) Dynamic Traffic Assignment [6]:

In the Merchant and Nemhauser model of dynamic traffic assignment [10], a traffic network is represented by a directed graph. One of the nodes is designated as the destination. The planning horizon is divided into a finite number of discrete time periods. For each time period, external inputs are allowed at any node except the destination. For each arc, there is an exit function which relates the amount of traffic entering and leaving the arc during a time period. Congestion is modeled by assuming the exit functions to be nondecreasing, continuous, piecewise linear and concave. The problem is to find the feasible traffic flow which minimizes the total amount of traffic over the planning horizon.

Here the unknowns are the amount of traffic in each arc in each time period. They are transformed to convex combinations of the grid points for the piecewise linear exit functions. Therefore, the variables x_t are the interpolation weights for these combinations. The single-period constraints are the flow balance equations for the nodes. The dynamic constraints are the flow balance equations for the arcs.

The Staircase algorithm is used as part of a hybrid algorithm [6] for this class of problems. See Table 4.

9. Remarks

It has been observed in [3] that relative to a direct simplex approach, the Staircase algorithm tends to become more efficient with increasing problem size. However, the threshold problem size differs considerably for different classes of staircase problems. The results presented here, though favoring decomposition in every case, simply substantiate that observation. They should not be interpreted as measures of the relative performance of the two approaches in terms of absolute problem size.

ACKNOWLEDGEMENT

Part of this paper is based on the author's doctoral dissertation at Stanford University. The author is indebted to Alan Manne, George Dantzig and John Tomlin for their guidance and encouragement.

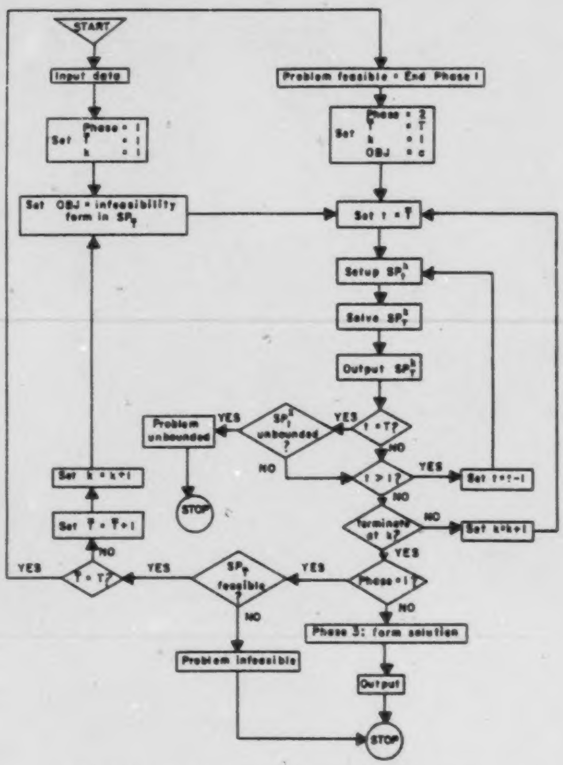

START
Input data
Set Phase = 1
$\bar{T}$ = 1
k = 1
Set OBJ = infeasibility form in $SP_{\bar{T}}$
Problem feasible = End Phase I
Set Phase = 2
$\bar{T}$ = T
k = 1
OBJ = c
Set t = $\bar{T}$
Setup SP_t^k
Solve SP_t^k
Output SP_t^k
t = T?
YES
SP_t^k unbounded ?
YES
Problem unbounded
STOP
NO
NO
t > 1?
YES
Set t=t-1
NO
Terminate at k?
NO
Set k=k+1
YES
Phase = 1?
YES
$SP_{\bar{T}}$ feasible ?
YES
$\bar{T}$ = T?
YES
NO
Set $\bar{T}$ = $\bar{T}$+1
Set k = k+1
NO
Problem infeasible
NO
Phase 3: form solution
Output
STOP

Figure 1. Flow Diagram of the Staircase Algorithm

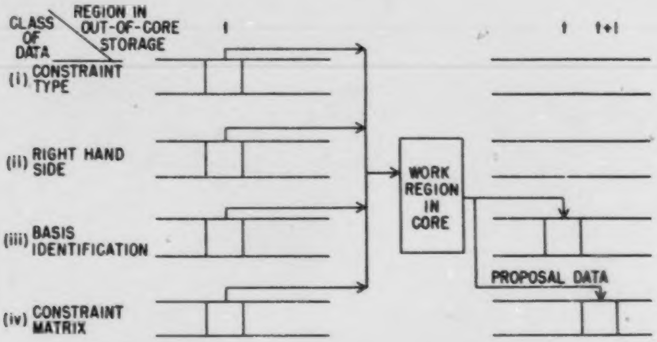

CLASS OF DATA
REGION IN OUT-OF-CORE STORAGE
t
t
t+1
(i) CONSTRAINT TYPE
(ii) RIGHT HAND SIDE
(iii) BASIS IDENTIFICATION
(iv) CONSTRAINT MATRIX
WORK REGION IN CORE
PROPOSAL DATA

Figure 2. Data Flow for a Subproblem in the Staircase Algorithm

NAME	components	dimensions
x	(s_t, x_t, λ_t)	$(m_t + 1 + m_{t+1} + n_t + k) \times 1$
c	$(0, c_t, p_t)$	$(m_t + 1 + m_{t+1} + n_t + k) \times 1$
A	$\begin{bmatrix} I & 0 & A_t & Q_t \\ 0 & 1 & 0 & b_t \end{bmatrix}$	$(m_t + 1) \times (m_t + 1 + m_{t+1} + n_t + k)$
B	$[0 \;\; I \;\; B_t \;\; 0]$	$m_{t+1} \times (m_t + 1 + m_{t+1} + n_t + k)$
M	$\begin{bmatrix} 1 & c \\ 0 & A \\ 0 & B \end{bmatrix}$	$(1 + m_t + 1 + m_{t+1}) \times (1 + m_t + 1 + m_{t+1} + n_t + k) = m \times n$
d	$(0, d_t, 1, 0)$	$(1 + m_t + 1 + m_{t+1}) \times 1$
$\widehat{NAME}$	a subset of those of NAME	e.g., $\hat{M}$ is $(1 + m_t + 1 + m_{t+1}) \times (1 + m_t + 1 + m_{t+1})$

Table 1. Notations for a Full Subproblem

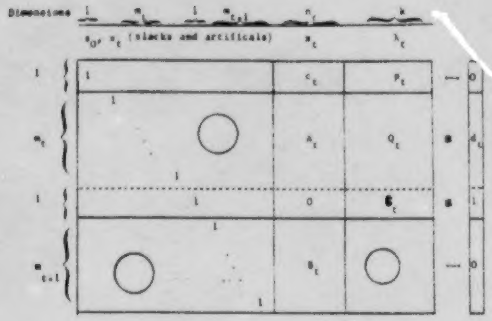


Fig. 3. A Full Subproblem

STATISTICS \ PROBLEM		SCSD1	SCSD2	SCSD3	SCSD4	SCSD5
LP	PERIODS	3	3	4	6	20
	ROWS	78	78	148	102	408
	COLUMNS	838	838	2048	834	4228
	NONZEROS	3226	3226	8288	3112	16604
	% DENSITY	4.94	4.94	2.73	3.66	0.96
MPS/360	CPU SECONDS	6.3	8.7	33.9	9.7	>120.
SC73	CPU SECONDS	4.1	4.9	12.7	3.6	41.6

Table 2. Statistics of the Structural Design Problems

STATISTICS \ PROBLEM		SCRS8			
		1A	2A	3A	4A
LP	PERIODS	4	4	4	4
	ROWS	491	491	491	491
	COLUMNS	1538	1633	1649	1660
	NONZEROS	4433	4460	4476	4520
	% DENSITY	0.55	0.56	0.55	0.55
LPM1	CPU SECONDS	30.8	26.7	30.3	40.1
SC74	CPU SECONDS	20.8	19.2	18.5	19.4

Table 3. Statistics of the Dynamic Energy Model Problems

STATISTICS \ PROBLEM		SCTAP1	SCTAP2	SCTAP3
LP	PERIODS	10	10	10
	ROWS	311	1101	1491
	COLUMNS	791	2981	3971
	NONZEROS	3683	14395	19045
	% DENSITY	1.50	0.44	0.32
LPM1	CPU SECONDS	7.3	103.8	223.3
SC74	CPU SECONDS	3.8	15.6	17.7

Table 4. Statistics of the Dynamic Traffic Assignment Problems

REFERENCES

[1] Beale, E. M. L., "Sparseness in linear programming", in Large Sparse Sets of Linear Equations, ed., J. K. Reid, Academic Press, London, 1971, pp. 1-15.

[2] Dantzig, G. B., and P. Wolfe, "Decomposition principle for linear programs", Operations Research 8, 1960, pp. 101-111.

[3] Ho, J. K., "Nested decomposition of large scale linear programs with the staircase structure", Doctoral dissertation, Stanford University, California, 1974. (Also Technical Report 74-4, Systems Optimization Laboratory, Department of Operations Research, Stanford University, May, 1974.)

[4] Ho, J. K., "Optimal design of multistage structures: a nested decomposition approach", Computers and Structures 5, 1975, pp. 249-255.

[5] Ho, J. K., "Nested decomposition of a dynamic energy model", Technical Report AMD 705, Brookhaven National Laboratory, New York, June, 1975. (To appear in Management Science)

[6] Ho, J. K., "A hybrid algorithm for the dynamic traffic assignment problem", Technical Report AMD 731, Brookhaven National Laboratory, New York, March, 1976.

[7] Ho, J. K., and A. S. Manne, "Nested decomposition for dynamic models", Mathematical Programming 6 (1974) 121-140.

[8] Kalan, J. E., "Aspects of large-scale in-core linear programming", in Proceedings of A.C.M. Annual Conference, 1971, pp. 304-313.

[9] Manne, A. S., "U.S. Options for a transition from oil and gas to synthetic fuels", Discussion Paper No. 26D, Public Policy Program, John F. Kennedy School of Government, Harvard University, January, 1975.

[10] Merchant, D. K., and G. L. Nemhauser, "A model and an algorithm for the dynamic traffic assignment problem", Technical Report No. 247, Department of Operations Research, Cornell University, New York, January, 1975.

[11] Orchard-Hays, W., Advance Linear Programming Computing Techniques, McGraw-Hill Book Company, New York, 1968.

[12] Pooch, U. W., and A. Nieder, "A survey of indexing techniques for sparse matrices", Computing Survey 5, 1973, 109-133.

[13] Simonnard, M., Linear Programming, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1966.

[14] Tomlin, J. A., "Pivoting for size and sparsity in linear programming inversion routines", Journal of the Institute of Mathematics and its Applications 10, 1972, pp. 289-295.

[15] Tomlin, J. A., "LPM1 user's manual", Systems Optimization Laboratory, Department of Operations Research, Stanford University, 1973.

A Stepping-stone Parallel-cut Method for Integer Programming

by To-Yat Cheung
Department of Computer Science
University of Ottawa
Ottawa, Ontario, Canada

ABSTRACT

This paper presents a cutting plane method for integer programming. Let $\bar{a}_0$ be the optimal point of the associated linear program and $\bar{n}$ be the normal to the objective function hyperplane at $\bar{a}_0$. On the halflines of the cone incident at $\bar{a}_0$, we determine those points where the halflines intersect the coordinate planes and then project them onto $\bar{n}$. With the projections as stepping-stones, the objective function hyperplane (the parallel cut) is pushed into the cone step by step. At each step, a minimum number of integer points is generated for feasibility and optimality tests. The first feasible integer point 'trapped' by the parallel cut is the optimal point of the integer program.

Other main features of this method are: (1) In practice, it is not necessary to compute the cut. (2) In general, the candidates generated at each step give better value to the objective function than the candidates generated at the next step. (3) Reoptimization is not required.

Keywords: Integer programming, cutting plane, cone, normal, projections.

1. INTRODUCTION

This paper presents a cutting plane method for integer programming. The cut is always parallel to the hyperplane of the objective function (called a parallel cut). Let $\bar{a}_0$ be the optimal point of the associated linear program and $\bar{n}$ be the normal to the objective function hyperplane at $\bar{a}_0$. On the halflines of the cone incident at $\bar{a}_0$, we determine those points where the halflines intersect the coordinate planes. The intersection points are projected onto $\bar{n}$. These projections are then used as stepping-stones for moving the parallel cut into the cone step by step. At each step, the intersections on the halflines are used to generate a set of integer points for feasibility and optimality tests. The process terminates as soon as a feasible integer point is 'trapped' by the parallel cut.

Our parallel-cut method is closely related to three other approaches developed in the last few years, namely the methods of convexity cuts [1,2,3,8,11,12], enumerative cuts [4,5,6], and cut-search [7,8,9]. However, these approaches have some or all of the following disadvantages:

1. There exist no criteria for the choice of convex regions (in the case of convexity cuts and enumerative cuts) or for the choice of halflines (in the case of cut-search).

2. Extra computational work is required to generate the cuts. The amount of work depends mainly on the convex region used.

3. A cut may be either very shallow or in the wrong 'inclination', i.e. too many bad candidates may be generated.

4. Reoptimization is required.

5. Usually, even after a feasible integer point has been 'trapped' by a cut, search for a better solution still has to be continued.

In contrast, our parallel-cut algorithm has the following main features:

1. No choice of convex regions or edges is required. A parallel cut is generated step by step in a definite manner.

2. A parallel cut is merely a conceptual cut. In practice, no computation is involved in its generation.

3. The direction of a parallel cut is steepest and hence may be regarded as the proper 'inclination'. Together with stepsize control, a parallel cut generates, at each step, a minimum number of good candidates.

4. Reoptimization is not required.

5. Once a feasible integer point is 'trapped' by a cut, it is optimal and the process terminates.

Parallel cuts are also used, conceptually, in Hillier's bound-and-scan algorithm [10], but in a different manner.

2. GEOMETRIC INTERPRETATION OF THE PARALLEL-CUT METHOD

In this section, we illustrate by an example

43

the geometrical motivation of our parallel-cut method. For expository purposes, we shall use the following definitions.

Definition (O-plane, $O(x^*)$-plane) An O-plane is a hyperplane of the form $cx=\alpha$, where cx is the objective function and α is a constant. In particular, an $O(x^*)$-plane has the form $cx=cx^*$, where x^* is a fixed point on the plane.

In Figure 1, C_1 and C_2 are two halflines incident at $\bar{a}_0$ and $\bar{n}$ is the (negative) normal to the $O(\bar{a}_0)$-plane at $\bar{a}_0$. We observe that the coordinate planes through an integer point inside the cone intersects at least one of the halflines. For example, the coordinate planes through I_3 intersect C_1 and C_2 at H_1^2 and H_2^1 respectively. Hence, one of the coordinates of H_1^2 (and H_2^1) must be integral.

Definition (H-point) An H-point is a point on a halfline incident at $\bar{a}_0$ such that at least one of its coordinates has integral value.

Starting from $\bar{a}_0$, let us move the O-plane in the direction of $\bar{n}$. As it moves forward, it will 'trap' infinitely many integer points, some lying outside the cone (e.g. I_1), some lying inside the cone but violating certain nonbinding constraints (e.g. I_2). The first feasible integer point (here I_3) 'trapped' is obviously the optimal point. Note that if these integer points are projected perpendicularly onto $\bar{n}$, the image of the optimal integer point has the shortest distance from $\bar{a}_0$ among all feasible integer points.

Using the projections p_0, p_1, p_2, p_3, p_4, and p_5 as stepping-stones, Table 1 shows the H-points, integer coordinates and integer points newly generated at each step. Note that, at p_5, I_3 becomes optimal, because it is the first feasible integer point 'trapped'.

p_i	H-points	integer coordinates	integer points
p_0	none	none	none
p_1	H_2^1	$x_1=2$	none
p_2	H_2^2	$x_1=1$	none
p_3	H_2^3	$x_2=1$	I_2 (infeasible), I_3 (curr. optimal)
p_4	H_1^1	$x_1=3$	I_4 (rejected, worse than I_3)
p_5	H_1^2	none	none

Table 1

Information newly generated at each step in Fig. 1.

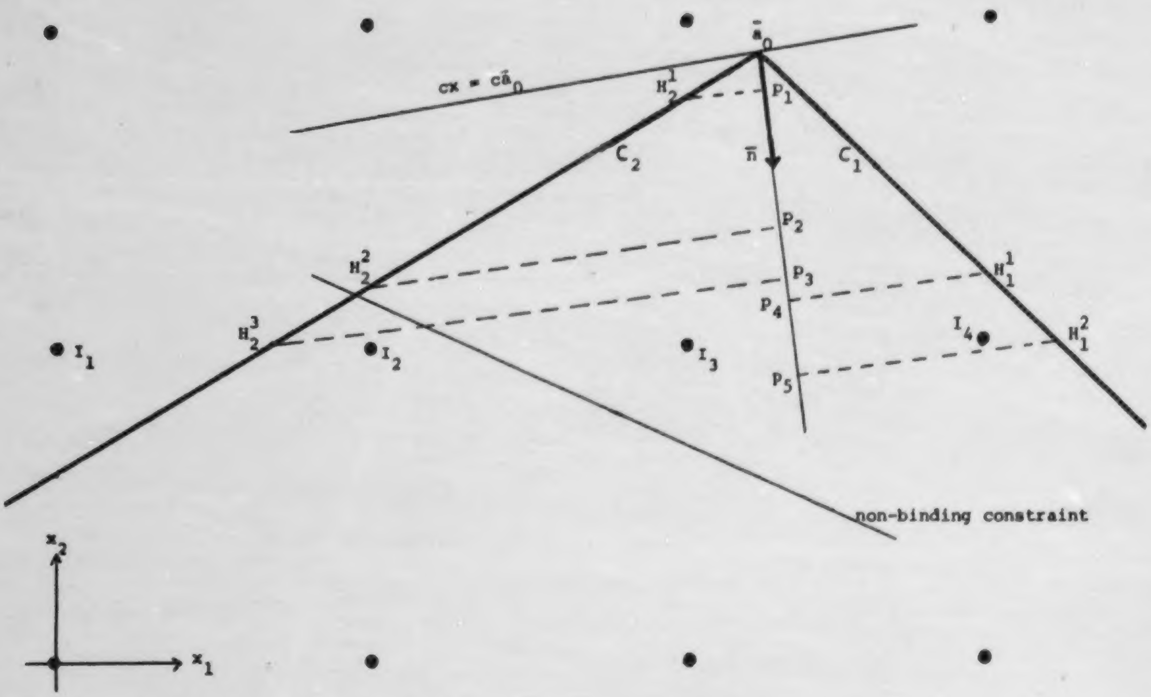


Figure 1 Illustration of the parallel-cut method.

3. PROBLEM FORMULATION AND MATHEMATICAL PRELIMINARIES

Consider the pure integer linear program

$$Q: \begin{cases} \text{maximize} & cx \\ \text{subject to} & Ax \leq b \\ & x_i \geq 0, \text{ integer}, i \in N, \end{cases} \qquad (3.1)$$

where $N \equiv \{1,2,\ldots,n\}$, c, x are n-vectors, A is an $m \times n$ matrix and b is an m-vector. Throughout this paper, we refer to x_i, $i \in N$, as the structural variables.

The associated linear program Q' is obtained from the integer linear program Q by dropping the integrality requirement. Introducing m slack variables $s = \text{col}(s_1, s_2, \ldots, s_m)$, Q' can be written in the standard form

$$Q'': \begin{cases} \text{maximize} & x_0 = cx \\ \text{subject to} & Ax + s = b \\ & x_i \geq 0,\ i \in N,\ s_j \geq 0,\ j \in M, \end{cases} \qquad (3.2)$$

where $M \equiv \{1,2,\ldots,m\}$.

Suppose Q" has an optimal solution. Let B and J be the index sets of the basic structural and basic slack variables respectively and y_{ij} be the coefficients of the optimal simplex tableau. Then the objective function value and each of the basic structural variables can be expressed in terms of the nonbasic (structural and slack) variables t_j as

$$x_0 = y_{00} - \Sigma_{j \in N} y_{0j} t_j$$

$$x_i = y_{i0} - \Sigma_{j \in N} y_{ij} t_j, \ i \in B. \qquad (3.3)$$

Let us replace every y_{ij} in (3.3) by another notation $\bar{a}_{ij}$, for $i \in B$ and $j \in \{0\} \cup N$. Attaching, for each nonbasic structural variable x_i, the trivial relation setting x_i equal to itself, i.e. a relation of the form (3.3) in which $\bar{a}_{i0} = 0$, $\bar{a}_{ii} = -1$ and $\bar{a}_{ij} = 0$ for $i \neq j$, $i \in N-B$, and rewriting in vector notation, we obtain the following linear program over a cone

$$C: \begin{cases} \text{maximize} & x_0 = y_{00} - \Sigma_{j \in N} y_{0j} t_j \\ \text{subject to} & x = \bar{a}_0 - \Sigma_{j \in N} \bar{a}_j t_j \geq 0,\ t_j \geq 0 \\ & x_i \text{ integer}, i \in N. \end{cases} \qquad (3.4)$$

The fundamental relation between the problems C and Q is as follows: Let t* be an optimal solution to C and $x^* \equiv \bar{a}_0 - \Sigma_{j \in N} \bar{a}_j t_j^*$. Then x* is an optimal solution to Q if and only if x* also satisfies those constraints of Q which are not binding at $\bar{a}_0$.

The j^{th} halfline of the cone (3.4) is defined by

$$C_j: \xi_j = \bar{a}_0 - \bar{a}_j t_j, \ t_j \geq 0. \qquad (3.5)$$

The quantities $\bar{a}_0$, $\bar{a}_j$ and $c\bar{a}_j$, $j \in N$, will be used in Section 4. The next lemma shows that they can be obtained from the optimal simplex tableau of the linear program Q".

Lemma 1 Suppose the following $m \times (n+1)$ matrix

$$\begin{bmatrix} Y_1 & Y_2 \\ Y_3 & Y_4 \end{bmatrix} \begin{array}{l} \text{basic structural variables} \\ \text{basic slack variables} \end{array}$$

is obtained from the optimal simplex tableau of Q", where Y_1 and Y_3 correspond to the columns of the current optimal point and the nonbasic slack variables, whereas Y_2 and Y_4 correspond to the columns of the nonbasic structural variables. Then the column vectors $\bar{a}_j$ of (3.4) are the columns of the following $n \times (n+1)$ matrix

$$\begin{bmatrix} Y_1 & Y_2 \\ 0 & -I \end{bmatrix} \begin{array}{l} \text{basic structural variables} \\ \text{nonbasic structural variables,} \end{array}$$

where I is an identity matrix.

Furthermore, suppose y_{0j}, $j \in N$, are the relative-cost factors obtained from the optimal simplex tableau. Then

$$c\bar{a}_j = -y_{0j}, \ j \in N. \qquad (3.6)$$

Proof. The first part of the lemma follows from the definition of $\bar{a}_j$. To derive (3.6), we see that, for every t_j, $j \in N$, $y_{0j} \equiv \delta_j - \Sigma_{i \in J} 0 y_{ij} - \Sigma_{i \in B} c_i y_{ij} = -\Sigma_{i \in \bar{B}} c_i \bar{a}_{ij} - \Sigma_{i \in B} c_i \bar{a}_{ij} = -c\bar{a}_j$, where δ_j is equal to the cost associated with t_j if t_j is a structural variable, and is equal to 0 if t_j is a slack variable, and $\bar{B} = N-B$. (Q.E.D.)

4. THE PARALLEL-CUT ALGORITHM

In this section, we develop the parallel-cut algorithm first under the assumption that the optimal solution of the associated linear program Q' is unique and non-degenerate. Non-uniqueness and degeneracy will be discussed in Section 4.6.

The parallel-cut algorithm includes the following operations:

1. Generate H-points on the halflines.
2. Project the H-points onto the normal $\bar{n}$.
3. Using the projections as stepping-stones, generate a set of candidates at each step for feasibility and optimality tests.

The details of these operations are described in the next four subsections.

4.1 GENERATING H-POINTS ON THE HALFLINES

The use of H-points was first developed by Glover [7,8]. However, he used them for generating cuts directly; whereas we use them for generating projections and candidates. Let us first restate without proof one of his lemmas, using our notations.

Lemma 2 (First cut-search lemma of Glover)

Assume x' is contained in the truncated cone of points satisfying both (3.4) and

$$\Sigma_{j \in N} (1/t_j^*) t_j \leq 1, \qquad (4.1)$$

where $t_j^* > 0$ for all $j \in N$. Then every hyperplane $L(x-x') = 0$ through x' (for L a non-zero row vector) intersects at least one of the edges of the truncated cone incident at $\bar{a}_0$ (i.e. the line segments $x=\bar{a}_0-\bar{a}_j t_j$, $t_j^* \geq t_j \geq 0$).

In particular, let x' be an integer point inside the cone (3.4), $t_j^*=\infty$ for every $j \in N$, and $L=(0,0,\ldots,0,1,\ldots,0)$, where 1 is in the ith component, $i=1,2,\ldots,n$. Then, Lemma 2 implies that every coordinate plane through an integer point in the cone intersects at least one of its halflines. The H-points generated on the jth halfline are

$$H_j^\mu: \quad \xi_j^\mu=\bar{a}_0-\bar{a}_j t_j^\mu, \tag{4.2}$$

where the increasing sequence of parametric values of t_j are defined as follows:

$$t_j^1=\min\{t_j \mid t_j \geq 0 \text{ and } \bar{a}_{i0}-\bar{a}_{ij}t_j \text{ is a nonnegative integer for some } i\}$$

$$t_j^{\mu+1}=\min\{t_j \mid t_j > t_j^\mu \text{ and } \bar{a}_{i0}-\bar{a}_{ij}t_j \text{ is a nonnegative integer for some } i\}.$$

If there is no value of t_j that can be generated in this way, the t_j^1 or t_j^μ is defined to be equal to ∞. Numerically, the values of t_j^μ satisfying the above definition may be obtained as follows:

$$t_j^0=0, \quad t_j^{\mu+1}=t_j^\mu+\min_{i \in N}\{\phi_i\}, \quad \mu=0,1,2,\ldots, \tag{4.3}$$

where

$$\phi_i=\begin{cases} ((\bar{a}_{i0}-\bar{a}_{ij}t_j^\mu)-\langle\bar{a}_{i0}-\bar{a}_{ij}t_j^\mu\rangle)/\bar{a}_{ij} & \text{if } \bar{a}_{ij}<0 \\ ((\bar{a}_{i0}-\bar{a}_{ij}t_j^\mu)-\{\bar{a}_{i0}-\bar{a}_{ij}t_j^\mu\})/\bar{a}_{ij} & \text{if } \bar{a}_{ij}>0 \\ \infty & \text{otherwise,} \end{cases}$$

$\langle z\rangle$ denotes the smallest integer greater than z,

and $\{z\}$ denotes the largest integer smaller than z.

4.2 PROJECTING THE H-POINTS ONTO THE NORMAL

Let $\bar{n}$ be the (negative) normal to the $O(\bar{a}_0)$-plane at $\bar{a}_0$. The distance of any point x from the $O(\bar{a}_0)$-plane is given by

$$d(x) \equiv c(\bar{a}_0-x)/\|c\|, \tag{4.4}$$

where $\|\cdot\|$ denotes the Euclidean norm. In particular, the distance between $\bar{a}_0$ and the projection of an H-point (4.2) on $\bar{n}$ is given by (see Lemma 1).

$$d(\xi_j^\mu)=(c\bar{a}_j/\|c\|)t_j^\mu=(-y_{0j}/\|c\|)t_j^\mu, \quad j \in N. \tag{4.5}$$

Note that the quantities $y_{0j} \leq 0$ can be obtained from the optimal simplex tableau of the problem Q". If $\bar{a}_0$ is the unique optimal point, we have $y_{0j}<0$ for every $j \in N$.

Let us denote these projections by an increasing sequence

$$P=\{p_0,p_1,p_2,\ldots,p_k,\ldots\}, \tag{4.6}$$

where $p_0=0$, $p_i<p_{i+1}$. When there is no confusion, p_i is used to denote both a projection image and its distance from $\bar{a}_0$ along $\bar{n}$.

4.3 SETS OF CANDIDATE INTEGER POINTS

In this subsection, we describe how (4.6) is used to generate integer points for feasibility and optimality tests. Every $p_k \in P$ corresponds to one or several H-points, each of which in turn corresponds to one or several integer values of the coordinates x_i's. Hence, for each p_k, n discrete sets of values of the coordinate x_i

$$X_i(k)=\{x_i^1,x_i^2,\ldots,x_i^\nu\}, \quad i \in N \tag{4.7}$$

can be uniquely determined. (ν depends on i and k.) Candidate integer points are then obtained by forming all the possible combinations as follows:

$$(x_1,x_2,\ldots,x_i,\ldots,x_n), \quad x_i \in X_i(k), \quad i \in N.$$

The following lemma shows that each of the discrete sets $X_i(k)$ consists of consecutive integers without gaps.

Lemma 3 The discrete set $X_i(k)$ can be expressed in the form

$$X_i(k) \equiv \{x_i \mid \ell_i(k) \leq x_i \leq u_i(k), \ x_i \text{ integer}\},$$

where $\ell_i(k)$ and $u_i(k)$ are non-negative integer bounds.

Proof. Without loss of generality, we may assume that the elements of $X_i(k)$ satisfy $0 \leq x_i^1<x_i^2<\ldots<x_i^q<x_i^{q+1}<\ldots<x_i^\nu$. Suppose there is a gap of size g (≥ 2) between x_i^q and x_i^{q+1}, i.e. $x_i^{q+1}=x_i^q+g$. Let x_i^q and x_i^{q+1} correspond to two H-points on the halflines C_α and C_β respectively (the possibility that $\alpha=\beta$ is not excluded), i.e. there exist $t_\alpha^*>0$ and $t_\beta'>0$ such that

$$x_i^q=\bar{a}_{i0}-\bar{a}_{i\alpha}t_\alpha^*, \quad x_i^{q+1}=\bar{a}_{i0}-\bar{a}_{i\beta}t_\beta'. \tag{4.8}$$

We distinguish between two cases: (1) $x_i^{q+1}>\bar{a}_{i0}+1$; and (2) $x_i^{q+1} \leq \bar{a}_{i0}+1$. For the case (1), (4.8) implies that $-\bar{a}_{i\beta}t_\beta'>1$. Define: $t_\beta^o \equiv t_\beta'+1/\bar{a}_{i\beta}$ and $x_i^o \equiv \bar{a}_{i0}-\bar{a}_{i\beta}t_\beta^o$. It follows easily that $t_\beta'>t_\beta^o>0$ and $x_i^{q+1}>x_i^o=x_i^{q+1}-1>x_i^{q+1}-g=x_i^q$. This means that, on C_β, we can find an H-point $\bar{a}_0-\bar{a}_\beta t_\beta^o$ (in front of the H-point $\bar{a}_0-\bar{a}_\beta t_\beta'$) which generates the integer x_i^o between x_i^q and x_i^{q+1} — a contradiction. Similarly for the case (2), we can find on C_α an H-point which generates an integer between x_i^q and x_i^{q+1}. (Q.E.D.)

As a result of Lemma 3, it is computationally much easier to generate candidates. Instead of (4.7), we simply keep track of a pair of bounds $\ell_i(k)$ and $u_i(k)$ for every x_i, $i \in N$, and every p_k, $k=0,1,2,\ldots$, using the following stipulation:

$$\text{(i)} \quad \ell_i(-1)=\lceil\bar{a}_{i0}\rceil, \quad u_i(-1)=\lfloor\bar{a}_{i0}\rfloor. \tag{4.9}$$

(ii) Let $S_i(k)$ be the set of integral x_i

values associated with those H-points which are projected onto p_k. Then

$$\ell_i(k)=\begin{cases}\min\limits_{x_i\in S_i(k)} \{\ell_i(k-1),x_i\} & \text{if } S_i(k) \text{ is not empty,}\\ \ell_i(k-1) & \text{if } S_i(k) \text{ is empty,}\end{cases} \tag{4.10}$$

$$u_i(k)=\begin{cases}\max\limits_{x_i\in S_i(k)} \{u_i(k-1),x_i\} & \text{if } S_i(k) \text{ is not empty}\\ u_i(k-1) & \text{if } S_i(k) \text{ is empty,}\end{cases} \tag{4.11}$$

where $\lfloor z\rfloor$ (or $\lceil z\rceil$) is the largest (or smallest) integer which is smaller (or greater) than or equal to z. Note that there exists, for each k, at least one i such that either $\ell_i(k)<\ell_i(k-1)$ or $u_i(k)>u_i(k-1)$.

Hence, the set of candidates generated by $\{p_0,p_1,\dots,p_k\}$, is of the form

$$I(k)=\{x \mid \ell_i(k)\le x_i\le u_i(k),\ x_i \text{ integer},\ i\in N\}. \tag{4.12}$$

4.4 FEASIBILITY AND OPTIMALITY TESTS

At the projection p_k, our subproblem is

$$\begin{cases}\text{maximize} & x_0=cx\\ \text{subject to} & Ax\le b,\ x\in I(k).\end{cases} \tag{4.13}$$

A direct or algorithmic search may be required to determine whether (4.13) is infeasible or has an optimal solution. If infeasible, we proceed to the next projection p_{k+1}. Otherwise, (4.13) has an optimal solution. One of the main features of our parallel-cut algorithm is provided by the next result.

__Theorem 4__ Suppose x(k) is an optimal point of (4.13). If $d(x(k))\le p_k$, then x(k) is also an optimal point of the integer linear program Q.

Proof. We shall prove this theorem by showing the contradiction that, if x' is a feasible point of Q satisfying $cx(k)<cx'$. Then $x'\in I(k)$.

Consider the halfspace associated with the $O(p_k)$-plane

$$\Sigma_{j\in N}(1/t_j^*)t_j\le 1,\ t_j\ge 0,\ j\in N, \tag{4.14}$$

where $t_j^*=p_k\|c\|/(-y_{0j})>0$, $j\in N$. (See (4.5) and [1].) Suppose there exists a feasible point $x'\equiv\bar{a}_0-\Sigma_{j\in N}\bar{a}_j t_j'$ of Q such that $cx(k)<cx'$. Then, by (4.4), we have $d(x')=c(\bar{a}_0-x')/\|c\|=c([\bar{a}_0-x(k)]+[x(k)-x'])/\|c\|<d(x(k))\le p_k$ and $t_j'\le d(x')\|c\|/(-y_{0j})<t_j^*$, $j\in N$. This implies that x' satisfies (4.14). By Lemma 2, every coordinate plane through x' intersects a halfline C_j at $\bar{a}_0-\bar{a}_j t_j^o$, say, for some $t_j^o\le t_j^*$. By definition of I(k), we have $x'\in I(k)$. (Q.E.D.)

4.5 DETAILED DESCRIPTION OF THE PARALLEL-CUT ALGORITHM.

A detailed description of the parallel-cut algorithm follows:

__Step 0__ Solve the associated linear program Q' by the simplex method and let $\bar{a}_0$ be its optimal point. If $\bar{a}_0$ is integral, it is also an optimal point of the integer program Q. Otherwise, go to Step 1.

__Step 1__ (Initialization)

1.1 Formulate the cone problem C. (see Lemma 1 of Section 3).

1.2 On each of the halflines C_j, $j\in N$, generate a sequence of H-points by (4.2) and (4.3). For each of the H-points, keep track of its integer coordinate(s).

1.3 By (4.5), project perpendicularly the H-points onto the normal $\bar{n}$ and arrange the sequence of projections in increasing order or magnitude

$$P=\{p_0,p_1,p_2,\dots,p_i,\dots\},$$

where $p_0=0$, $p_i<p_{i+1}$.

1.4 Let k=-1.

__Step 2__ (Moving the parallel cut into the cone)

2.1 Let k=k+1

2.2 Solve (4.13). If (4.13) is infeasible or has an optimal point x(k) such that $d(x(k))>p_k$, repeat Step 2. Otherwise, the optimal point x(k), for which $d(x(k))\le p_k$, is also an optimal solution of the integer program Q.

In practice, the following refinements may be incorporated into the above formal description of the algorithm.

Since the algorithm may terminate early in the process, it would be a waste of time to generate too many H-points and projections at the outset. Instead, we may specify intervals on $\bar{n}$ and the halflines and generate the H-points and projections within these intervals one after another only when they are needed. Suppose ℓ is the size of the interval on $\bar{n}$, then the corresponding interval size for t_j is $-\ell\|c\|/y_{0j}$, $j\in N$.

Also, Step 2 of the algorithm may be replaced by the following more elaborate one:

__Step 2'__ (Moving the parallel cut into the cone)

2.1' Find the smallest k, say k^o, such that the set $\{x \mid Ax\le b,\ x\in I(k^o)\}$ has at least one feasible point. (If such k does not exist, the integer program Q does not have a feasible solution.)

2.2' Solve (4.13) with $k=k^o$. Let x(k) be the optimal point.

2.3' If $d(x(k))\le p_k$, then x(k) is also an optimal point of Q. Otherwise, go to Step 2.4'.

2.4' Set k=k+1. Test whether the set

$$T(k)\equiv\{x \mid Ax\le b,\ \ x\in I(k)-I(k-1),\ cx>cx(k-1)\}$$

is feasible. If not, define x(k)=x(k-1) and go to Step 2.3'. Otherwise, go to Step 2.5'.

2.5' Solve the problem $\max\{cx \mid x\in T(k)\}$ by any direct or algorithmic searching method. Let x(k) be the optimal point. Go to Step 2.3'.

4.6 DEGENERACY, NON-UNIQUENESS AND FINITENESS

In this subsection, we show that the parallel-cut algorithm works without any modification when the optimal solution of the associated linear program Q' is degenerate. We also discuss the relation between the finiteness of the algorithm and the uniqueness of this optimal solution.

Suppose $(\bar{a}_0,\bar{s})$ is the optimal point of Q". Degeneracy occurs when some of the basic components of $\bar{a}_0$ or $\bar{s}$ have zero value. Define the index sets

$$B^O=\{i|\bar{a}_{i0} \text{ basic and } \bar{a}_{i0}=0\};$$

$$M^O=\{i|\bar{s}_i \text{ basic and } \bar{s}_i=0\}.$$

Geometrically, in the n-space of the structural variables, degeneracy implies that the hyperplanes associated with the halfspaces $\Sigma_{j\epsilon N}a_{ij}x_j\leq b_i$, $i\epsilon M^O$ or $x_i\geq 0$, $i\epsilon B^O$ also pass through $\bar{a}_0$. In the same space, let us define

$$X=\{x|\Sigma_{j\epsilon N}a_{ij}x_j\leq b_i,\ i\epsilon M;\ x_j\geq 0,\ j\epsilon N\}$$

and

$$X'=\{x|\Sigma_{j\epsilon N}a_{ij}x_j\leq b_i,\ i\epsilon M-M^O;\ x_j\geq 0,\ j\epsilon N-B^O\}.$$

Thus, X' is obtained from X by dropping the constraints associated with those variables (structural or slack) which are basic but have value 0. Obviously, $X\subset X'$. Balas [1] proves the following lemma.

Lemma 5 $\bar{a}_0$ is the vertex of X' and $cx=c\bar{a}_0$ is a supporting hyperplane for X'. X' has n distinct edges adjacent to $\bar{a}_0$, and each halfline (3.5) contains exactly one such edge.

Interpreted geometrically (see Figure 2), this lemma implies that, in case of degeneracy, some of the halflines (3.5) may not contain any edge of the cone (incident at $\bar{a}_0$) of the feasible region X of the problem Q'. However, each halfline (3.5) contains an edge of another polytope X'.

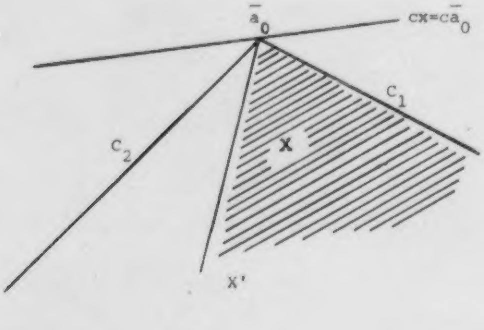


Figure 2 degeneracy at $\bar{a}_0$

In connection with our parallel-cut algorithm, it is obvious that the halflines (3.5) can also be used to generage H-points, which in turn are used to generate integer points to be tested for feasibility with respect to X. In other words, the algorithm works without any modification.

There is a close relation between the uniqueness of the optimal solution of the associated linear program Q' and the finiteness of the parallel-cut algorithm.

If the optimal point of Q' is unique, the following argument shows that the algorithm is finite. Let x' be an arbitrary feasible integer point (assumed existing) of Q. Then, the O(x')-plane intersects every halfline at a finite point and d(x') is an upper bound on the projections to be generated. Thus, the numbers of distinct H-points and projections are finite. Hence, the number of steps and the number of candidates to be tested at each step are both finite.

If some of the relative-cost factors y_{0j} are zero, the optimal point of Q' is not unique. Define the index set $D\equiv\{j|j\epsilon N,\ y_{0j}=0\}$ and the polytope

$$X"=\{x|x=\bar{a}_0-\Sigma_{j\epsilon D}\bar{a}_jt_j, t_j\geq 0,\ x\epsilon X\}.$$

Geometrically, non-uniqueness implies that X" lies on the $O(\bar{a}_0)$-plane (see Figure 3). Hence, all the H-points and integer points lying in X" are projected onto $\bar{a}_0$, and the discrete set I(0) as defined in (4.12) may not be empty. We distinguish between the following two cases:

(i) X is bounded.

For such a case, the number of H-points generated on each of the halflines C_j, $j\epsilon D$ is finite. By similar argument as in the uniqueness case, we can show the parallel-cut algorithm is finite if boundedness is incorporated into the definition (4.3).

(ii) X is unbounded.

For such a case, it may be necessary to generate infinitely many H-points on some of the C_j, $j\epsilon D$. If X" contains an integer point, Q becomes a problem of locating any such integer point. However, it may happen that, theoretically at least, X" does not contain any integer point but there exist feasible integer points of Q which are arbitrarily close to X". (See the remark below.) This implies that, at the initial projection p_0 of the algorithm, search for the best feasible integer point over the infinite set I(0) never terminates. In practice, such situation may be remedied by introducing an artificial constraint (the dotted line of Figure 3). Then an approximate solution is sought over a bounded portion of the unbounded feasible region.

Remark: It seems to the author that, theoretically speaking, this situation gives rise to trouble for most of the existing integer programming methods.

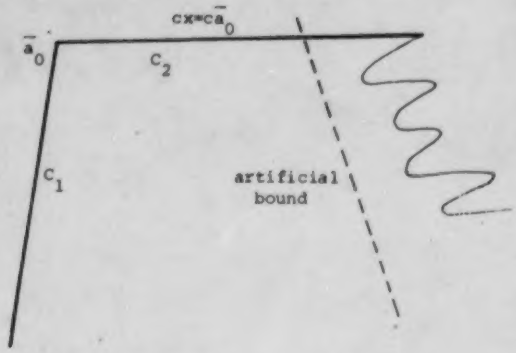


Figure 3 Non-uniqueness of optimal point

Application of the underlying ideas of the approach presented in this paper can obviously be extended to mixed integer programming, and probably to nonlinear integer programming with linear constraints.

REFERENCES

1. E. Balas, "Intersection cuts — a new type of cutting planes for integer programming", Oper. Res. 19 (1971) 19-39.

2. ________, "Integer programming and convex analysis: Intersection cuts from outer polars", Math. Programming 2 (1972) 330-382.

3. E. Balas, V.J. Bowman, F. Glover, D. Somer, "An Intersection cut from the dual of the unit hypercube", Oper. Res. 19 (1971) 40-44.

4. C.A. Burdet, "Enumerative cuts: I", Oper. Res. 21 (1973) 61-89.

5. ___________, "Convex and polaroid extensions", MS Report #73-21, Univ. of Ottawa, (1973).

6. ___________, "On the algebra and geometry of integer cuts", MS Report #74-8, Univ. of Ottawa, (1974).

7. F. Glover, "Cut search methods in integer programming", Math. Programming 3 (1972), 86-100.

8. _________, "Convexity cuts and cut search", Oper. Res. 21 (1973), 123-134.

9. _________, "Polyhedral annexation in mixed integer and combinatorial programming", Math. programming 8 (1975) 161-188.

10. F.S. Hillier, "A bound-and-scan algorithm for pure integer linear programming with general variables", Oper. Res. 17 (1969), 638-679.

11. H. Tui, "Concave programming under linear constraints", Doklady Akademii Nauk SSSR (1964), in Russian; English transl. Soviet Math. (1964) 1437-1440.

12. R.D. Young, "Hypercylindrically deduced cuts in zero-one integer programming", Oper. Res. 19 (1971) 1393-1405.

ACKNOWLEDGEMENT

This research was supported by the Canadian National Research Council Grant #A8963.

49

MONOMIAL PROGRAMMING*

T.L. Shaftel
The University of Arizona, Tucson and Queen's University, Kingston, Ontario

G.L. Thompson
Carnegie-Mellon University

Y. Smeers
Catholic University of Louvain

Abstract

A monomial programming problem is one of minimizing a polynomial in several variables subject to monomial constraints. A log transformation changes it into a problem with non-linear objective and linear constraints which, under certain conditions can be solved by Zangwill's convex simplex method. We show a direct method, based on our previous work, of solving the problem using a simplex-like tableau.

1. INTRODUCTION

In a recent series of papers [3, 4, 5, 6], we have presented simplex-like algorithms to several special kinds of problems arising in modular design, geometric programming, and elsewhere. In the present paper we shall specialize the general algorithm of [4] to the case of a programming problem in which the constraints involve monomial expressions. For this case a true simplex algorithm is possible in the sense that any of the standard linear programming routines can rather easily be modified to solve monomial constrained problems. Also all of the techniques for handling large scale linear programming problems, such as the revised simplex method, decomposition, lexicographic ordering, etc., are immediately transferable to the new algorithm. It therefore follows that monomially constrained problems in hundreds of variables can be solved.

In Sections 2-7 we present a description of the new method. Computational results are presented in Section 7. A monomial model concerning excess inventories is presented and solved in Section 8.

2. NOTATION

Because we will be working with monomial expressions and their derivatives in several variables we introduce some special vector and matrix notation to make it easy to do so. We define the $\odot$ product of an n-component row vector h and an n-component column vector x as:

$$(1)\quad h \odot x = (h_1, \ldots, h_n) \odot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = x_1^{h_1} x_2^{h_2} \cdots x_n^{h_n}$$

We extend this definition in the obvious way to the $\odot$ product of an mxn matrix H and an n-component column vector x as:

$$(2)\quad H \odot x = \begin{pmatrix} h_1 \\ \vdots \\ h_m \end{pmatrix} \odot x = \begin{pmatrix} h_1 \odot x \\ \vdots \\ h_m \odot x \end{pmatrix}$$

---

* This report was prepared as part of the activities of the Management Sciences Research Group, Carnegie-Mellon University, under Contract #N00014-67-A-0314-0007 NR 047-048 with the U.S. Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the U.S. Government.

Management Sciences Research Report #333, Management Sciences Research Group, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213.

where h_i is the ith row of H. As a numerical example observe that:

$$(3)\quad \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 3 \\ 0 & 0 & 2 \end{pmatrix} \odot \begin{pmatrix} 3 \\ 1 \\ 4/3 \end{pmatrix} = \begin{pmatrix} 9/4 \\ (4/3)^3 \\ (4/3)^2 \end{pmatrix}.$$

Using the $\odot$ product and the ordinary matrix product we can easily write a polynomial in several variables; let d be an m-component row vector, H an mxn matrix, and x an n-component column vector; then:

$$(4)\quad d(H \odot x) = d_1 x_1^{h_{11}} \dots x_n^{h_{1n}} + \dots$$
$$\dots + d_m x_1^{h_{m1}} \dots x_n^{h_{mn}}$$

is a polynomial in the variables.

We recall next the definition of the Schur product of vectors. Let a and b be two m-component row vectors; then the Schur product is defined as:

$$(5)\quad a*b = (a_1,\dots,a_m)*(b_1,\dots,b_m) = (a_1b_1,\dots,a_mb_m).$$

Using this we can write the derivative of a polynomial $d(H \odot x)$. Let $h^{(j)}$ be the jth column of H and let $h^{(j)'}$ be its transpose; let $H^{(j)}$ be the matrix H with 1 subtracted from each entry in the jth column. Since tne formula $\frac{dx^n}{dx} = nx^{n-1}$ works even when n=0, it follows that:

$$(6)\quad \frac{\partial}{\partial x_j}(d(H \odot x)) = (d*h^{(j)'})(H^{(j)} \odot x).$$

From this it further follows that:

$$(7)\quad x_j \frac{\partial}{\partial x_j}(d(H \odot x)) = (d*h^{(j)'})(H \odot x).$$

As an example, $f = 6x_1^1x_2^2x_3^3 + 7x_1^4x_2^5$ can be expressed in the form of equation (4) with $d = (6,7)$ and $H = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 0 \end{pmatrix}$. $\frac{\partial f}{\partial x_1} = 6x_2^2x_3^3 + 28x_1^3x_2^5$ and $x_1\frac{\partial f}{\partial x_1} = 6x_1^1x_2^2x_3^3 + 28x_1^4x_2^5$ can then be expressed as in equations (6) and (7) with $h^{(1)'} = (1,4)$ and $H^{(1)} = \begin{pmatrix} 0 & 2 & 3 \\ 3 & 5 & 0 \end{pmatrix}$.

3. PROBLEM STATEMENT

By a monomially constrained problem we shall mean a problem of the form

$$\begin{aligned} &\text{Minimize} \quad d(C \odot x) = g \\ (8)\quad &\text{Subject to} \quad A \odot x \geq b \\ &\qquad\qquad\qquad x \geq 0 \end{aligned}$$

where d is 1xk, c is kxn, A is mxn, x is nx1, and b is mx1. We assume b>0.

Note that by making the transformation:

$$(9)\quad x_j = e^{y_j}$$

and taking logarithms we could change (8) into the problem (using the obvious definitions):

$$(10)\quad \begin{aligned} &\text{Minimize} \quad d(C \odot e^y) \\ &\text{Subject to} \quad Ay \geq \ln b \end{aligned}$$

Problem (1) has linear constraints, but y is not constrained to be nonnegative. Following Charnes and Kirby [1] we then say that (8) is transformably linear.

The following facts are well known. If d>0 then $d(C \odot e^y)$ is a strictly convex function of $y_1,\dots,y_n$. If $d \geq 0$ then the same function is a convex (but not necessarily strictly convex) function of these variables.

If $d(C \odot e^y)$ is a convex function of y we could use Zangwill's convex simplex method [9] to solve problem (10) and thus (8). However in this paper we shall specialize our previous work [4] on simplex-like methods for solving nonlinear problems with nonlinear constraints to provide a simplex method that is somewhat more general than the convex simplex method for solving problem (8) directly.

Let $\lambda = (\lambda_1,\dots,\lambda_m)$ be an 1xm vector of Lagrange multipliers for the constraints of (8). Then the Lagrangian of (8) is:

$$(11)\quad L = d(C \odot x) - \lambda(A \odot x - b)$$

where $A \odot x = b$ now is considered to include the constraints x>0. Taking the partial derivative of L with respect to x_j, using (6), and setting the result equal to zero yields:

$$(12)\quad \frac{\partial L}{\partial x_j} = (d*c^{(j)'})(C^{(j)} \odot x) - (\lambda*a^{(j)'})(A^{(j)} \odot x) = 0.$$

Multiplying (12) by x_j and use (7) to obtain:

$$(13)\quad (\lambda*a^{(j)'})(A \odot x) = (d*c^{(j)'})(C \odot x).$$

We now use the well-known Kuhn-Tucker complementary slackness condition:

$$(14)\quad \lambda_i(a_{(i)} \odot x) = \lambda_i b_i \text{ for all } i$$

where $a_{(i)}$ is the ith row of A and define

$$(15)\quad \rho_i = \lambda_i b_i \text{ for all } i.$$

Substituting (14) and (15) into (13) gives:

$$(16)\quad \rho a^{(j)} = (d*c^{(j)'})(C \odot x) \text{ for all } j.$$

We shall call ρ_i the dual variables and (16) the dual equations.

As in linear programming duals associated with constraints of the type x>0 must be zero for basic variables so that at the optimum (16) must be satisfied for the original m constraints. If we let the right hand side of (16) be q_i for the basic variables then in matrix form $\rho B = q$ and $\rho = qB^{-1}$ where B is the basis. Likewise ignoring the nonnegativity constraints (16) becomes:

$$(d^* c^{(j)'})(C \odot x) - \rho a^{(j)} \leq 0 \quad \text{for}$$

nonbasic variables.
The left hand side of this inequality corresponds to the "reduced cost", r_j, values in linear programming, or explicitly:

$$\text{(16b)} \quad r_j = (d^* c^{(j)'})(C \odot x) - qB^{-1} a^{(j)} \quad \text{for all } j \text{ nonbasic,}$$

where $B^{-1}\ a^{(j)}$ is merely the jth column of the present tableau.

4. RELATIONSHIP TO LINEAR PROGRAMMING

Note the similarity between the ordinary linear programming dual equations and (16). If the ith constraint in (8) is not tight $\rho_i = 0$ and if it is tight $\rho_i \geq 0$. Hence we can use the ρ_i's or equivalently $\lambda_i = \frac{\rho_i}{b_i}$ in the same way as they are used in ordinary linear programming.

If we assume that one of the monomial constraints in (8) is tight and use it to eliminate a variable, say x_j, from the rest of the equations then it is not hard to see that this transformation will make linear changes in the exponents of the other monomials and log linear change in the right hand sides.

For instance, consider the polynomial equations:

$$x_1 x_2^2 = 3$$

$$x_1^2 x_2 = 9$$

$$x_1 x_2 = 4$$

Using the notation of the previous section these can be written $A \odot x = b$, or in detached coefficient form as:

$$\begin{array}{cc|c} x_1 & x_2 & \\ \hline 1 & 2 & 3 \\ 2 & 1 & 9 \\ 1 & 1 & 4 \end{array}$$

where A is the 3x2 matrix on the left and b is the 3x1 vector on the right. If we now want to use the second equation to eliminate x_1 from the other two equations we can multiply by the corresponding pivot matrix (see [2]) in front as follows:

$$\begin{pmatrix} 1 & -\frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & -\frac{1}{2} & 1 \end{pmatrix} \left[\begin{array}{cc|c} 1 & 2 & 3 \\ 2 & 1 & 9 \\ 1 & 1 & 4 \end{array}\right] = \left[\begin{array}{cc|c} 0 & \frac{2}{3} & 1 \\ 1 & \frac{1}{2} & 3 \\ 0 & \frac{1}{2} & \frac{4}{3} \end{array}\right]$$

Note that we used ordinary matrix multiplication in the A area of the tableau, but $\odot$ multiplication in the b-area. All the familiar rules of pivoting in linear programming apply if they are appropriately modified for operation on the right hand side.

Most of the problems encountered in solving ordinary linear programming problems such as degeneracy (which can be handled by perturbation techniques or lexiographic orderings) can occur, and some new difficulties as well. For instance it is possible that one of the factors of the objective function could tend to zero without a basis change being needed. To handle such problems we can employ regularization constraints of the form $(C \odot x)_i > k_i > 0$ where k_i is a small number. Most of the other difficulties can be handled by similar modifications of well known linear programming techniques and will not be discussed further here.

We next describe a simplex method for solving problems of the form (8). To start it we need a Phase I procedure. To this end we add slack variables z_i and artificial variables u_i to the constraints of (8) to obtain the equivalent problem:

$$\text{(17)} \quad \begin{cases} \text{Minimize} \quad d(C \odot x) \\ \text{Subject to} \\ \qquad (A,-I,I) \odot \begin{pmatrix} x \\ z \\ u \end{pmatrix} = b. \\ \qquad z_i, u_i, \geq 1 \end{cases}$$

Let f be a column vector of all ones.
If $b \geq f$ it follows that an initial feasible solution is $x=f$, $u=b$, so that the initial tableau for the simplex Phase I start is:

$$\text{(18)} \quad \begin{array}{|cc|c|} \multicolumn{1}{c}{x} & \multicolumn{1}{c}{z} & \multicolumn{1}{c}{} \\ \hline A & -I & b \\ \hline \end{array} \quad u\ .$$

If, however, some components of b are <1 then the initial feasible basis can be made up of the components of u_i for which $b_i > 1$ and the components of z_i for which $b_i \leq 1$. We now use a Phase I objective function $e \odot u$ (where e is an m-column vector of all ones) and pivot until all the u_i artificial variables have been eliminated in complete analogy to ordinary linear programming.

In the next section we describe in detail the simplex method for monomially constrained problems which is based on the general method described in our paper [4]. The main differences from the ordinary linear programming simplex method are that the variables are not constrained to be ≥ 1 and second that nonbasic variables can take on values other than 1 or 0. The latter is necessary in order that we can find optimun solutions that are not necessarily determined by intersections of constraining hyperplanes. We use this method of describing the algorithm rather than the parametric programming method of [4] because it seems to be simpler in the present context.

5. THE SIMPLEX ALGORITHM FOR MONOMIALLY CONSTRAINED PROBLEMS

The description of the algorithm is similar to that for ordinary linear programming. All of the variants of the latter can be made to the present algorithm without difficulty so we do not go into detail in their discussion here.

Phase I. Set up the problem, find the quantities, A, b, c, and d and substitute them into the initial tableau (18).

(A) Go to the MAIN routine using the objective function $g = u_1 u_2 \ldots u_m$. If the problem has a solution with value greater than 1 then the original problem has no solution; stop. If the problem has a solution with value 1 then all the artificial variables are nonbasic (or can easily be made nonbasic and eliminated from the tableau).

Phase II. Set $g = d(C \odot x)$ and go to the MAIN program. When that program is complete it will provide an answer to the originally stated problem, or to the regularized problem if infinite solutions are found.

MAIN program.

1. Dual Solution Routine
   - (a) For each basic variable v_i calculate the value of q_i at the current solution.
   - (b) For each nonbasic v_j calculate the reduced cost, r_j, using (16b) where the second term sumation is to be taken over the basic variables and hence the sum calculation can be done from the current tableau.
2. Find Incoming Vector.

   Calculate the sets

   $S_1 = \{v_j | v_j$ is non basic, v_j is tight at a lower bound and $r_j > 0.\}$

   $S_2 = \{v_j | v_j$ is non basic, v_j is not tight at a lower bound and $r_j \neq 0.\}$

   If $S_1 \cup S_2 = \emptyset$ the current solution is optimal.

   Go to 7. Else go to 3.
3. Find largest indicator.

   Find j as the index that maximizes

   $\text{Max}\{\underset{k \in S_1}{\text{Max}} |r_k|, \underset{k \in S_2}{\text{Max}} |r_k v_k|\}.$

   If $j \in S_1$ or $j \in S_2$ and $r_j > 0$ go to 4.

   If $j \in S_2$ and $r_j < 0$ go to 5.
4. Find solution with larger v_j.
   - (a) Find the maximum extent v_j^+ to which v_j can be increased while keeping feasibility. This can be done by using the column of the current tableau under v_j and the right hand side column. Determine outgoing vector v_h.
   - (b) Solve the auxiliary problem for variable v_j; let v_j^0 be its optimum value.
   - (c) If $v_j^+ \leq v_j^0$ go to 6.
   - (d) If $v_j^+ > v_j^0$, set $v_j = v_j^0$ in the tableau, calculate the modified right hand sides and go to 1.
5. Find solution with smaller v_j.
   - (a) Find the maximum extent v_j^- to which v_j can be decreased while keeping feasibility. This can be done by using the column of the current tableau under v_j and the right hand side column. Determine the outgoing vector v_h.
   - (b) Solve the auxiliary problem for variable v_j; let v_j^0 be its optimum value.
   - (c) If $v_j^- \geq v_j^0$ go to 6.
   - (d) If $v_j^- < v_j^0$, set $v_j = v_j^0$ in the tableau, calculate the modified right hand side and go to 1.
6. Pivot exchange: make v_j basic and v_h nonbasic. Let P be the corresponding pivot matrix (see [2]); then use the calculation $PA^{(old)} = A^{(new)}$ for the A part of the tableau and $P \odot b^{(old)} = b^{(new)}$ for the b part of the tableau
7. End of MAIN program.

AUXILIARY problem routine for nonbasic variable v_j.

1. Evaluate the objective function g at the current solution but assuming v_j is a variable. Let $g(v_j)$ be the corresponding function.
2. Use a search (or other) technique to find the unconstrained minimum value v_j^0 of $g(v_j)$.

6. EXAMPLES

We work two examples in order to illustrate the method. We state first the constraints:

$$\begin{aligned} & x_1 x_2^2 \geq 3 \\ & x_1^2 x_2 \geq 9 \\ (19)\quad & x_1 x_2 \geq 4 \\ & x_1, x_2 \geq 1 \end{aligned}$$

The feasible region is shown in Figure 7. Adding slack and artificial variables these become.

$$\begin{aligned} & x_1 x_2^2 z^{-1} u_1 = 3 \\ (20)\quad & x_1^2 x_2 z_2^{-1} u_2 = 9 \\ & x_1 x_2 z_3^{-1} u_3 = 4 \end{aligned}$$

The constraints $x_1 \geq 1$, $x_2 \geq 1$ will be imposed by the method in order to keep tableaus small. We now use the Phase I objective function $u_1 u_2 u_3$ and do the Phase I calculations in condensed tableau form (Figure 1). It is easy to show that $\frac{v_i \partial g_i}{\partial v_i} = u_1 u_2 u_3$ for $i = 1,2,3$. The corresponding dual varialbe calculations appear on the right and below the tableau. The pivot element is circled; the new tableau with variable u_1 dropped appears in Figure 2. The $\odot$ operation was performed on the right hand side. There is one positive indicator (reduced cost) in column 2. The second and third rows indicate the relationships $u_2 = x_2^3 z_1^{-2} z_2^1$ and $u_3 = (4/3) x_1 z_1^{-1} z_3$ so that when z_1 is decreased u_2 and u_3 are decreased. Since we don't want u_2 to go below 1 we pivot; the new tableau with variable u_2 dropped appears in Figure 3.

Using the same reasoning as before we pivot on the circled $\frac{1}{2}$ in Figure 3 and obtain the primal feasible tableau of Figure 4. The solution $x_1 = 9/4$, $x_2 = 16/9$, $z_1 = 64/27$, $z_2 = z_3 = 1$ and $u_1 = u_2 = u_3 = 1$ is feasible for the constraints (20).

Suppose now we wish to solve the problem:

$$\begin{aligned} & \text{Minimize } x_1^4 x_2 = g(x_1, x_2) \\ (21)\quad & \\ & \text{Subject to constraints (19).} \end{aligned}$$

The tableau of Figure 4 with dual variable calculations for this objective function is shown in Figure 5. If we increase z_3 we have $x_1 = (9/4) z_3^{-1}$ and $x_2 = (4/3)^2 z_3^2$ so that $g(z_3) = (9/4)^4 (4/3)^2 z_3^{-2}$ so that increasing z_3 decreases $g(z_3)$ indefinitely. But the limit on increases of z_3 is the requirement $x_1 \geq 1$. Therefore we pivot on the circled 1 in Figure 5 giving the tableau of Figure 6.

Hence the optimum solution is $x_1 = 1$, $x_2 = 9$, $z_1 = 27$, $z_2 = 1$, $z_3 = 9/4$ which can be seen in the graph of Figure 7. The value of the objective function is $g(x_1, x_2) = 9$.

We now solve a problem with the same constraints but a different objective function in order to show the solution of the auxiliary problem. The new problem is:

$$\begin{aligned} & \text{Minimize } x_1^4 x_2 + 2 x_1^{-1} x_2 \\ (22)\quad & \\ & \text{Subject of constraints (19).} \end{aligned}$$

	x_1	x_2	z_1	z_2	z_3			$q_i =$
	①	2	-1	0	0	3	$= u_1$	$u_1 u_2 u_3 = 108$
	2	1	0	-1	0	9	$= u_2$	$u_1 u_2 u_3 = 108$
	1	1	0	0	-1	4	$= u_3$	$u_1 u_2 u_3 = 108$
$r_j =$	432	432	-108	-108	-108			

Figure 1

$$
\begin{array}{c|cccc|ccc}
 & x_2 & z_1 & z_2 & z_3 & & & q_i = \\
\hline
 & 2 & -1 & 0 & 0 & 3 & x_1 & 0 \\
 & -3 & \textcircled{2} & -1 & 0 & 1 & u_2 & u_1u_2u_3 = 4/3 \\
 & -1 & 1 & 0 & -1 & 4/3 & u_3 & u_1u_2u_3 = 4/3 \\
\hline
r_j = & -16/3 & 4 & -4/3 & -4/3 & & &
\end{array}
$$

Figure 2

$$
\begin{array}{c|ccc|cc}
 & x_2 & z_2 & z_3 & & q_i = \\
\hline
 & \frac{1}{2} & -\frac{1}{2} & 0 & 3 = x_1 & 0 \\
 & -\frac{3}{2} & -\frac{1}{2} & 0 & 1 = z_1 & 0 \\
 & \textcircled{\frac{1}{2}} & \frac{1}{2} & -1 & \frac{4}{3} = u_3 & u_1u_2u_3 = \frac{4}{3} \\
\hline
r_j = & \frac{2}{3} & \frac{2}{3} & -\frac{4}{3} & &
\end{array}
$$

Figure 3

$$
\begin{array}{|cc|l}
z_2 & z_3 & \\
\hline
-1 & 1 & 9/4 = x_1 \\
1 & -3 & (4/3)^3 = z_1 \\
1 & -2 & (4/3)^2 = x_2 \\
\hline
\end{array}
$$

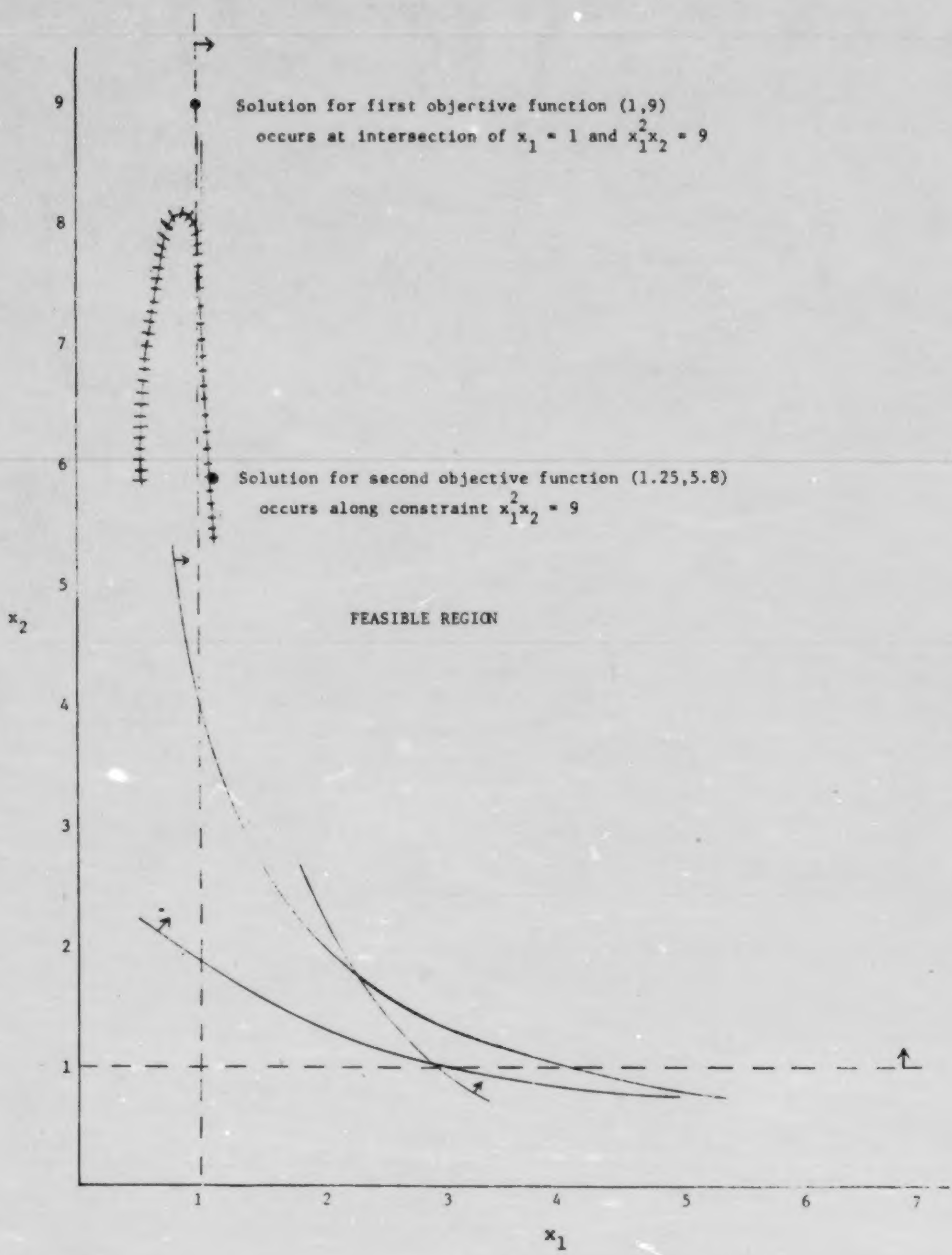
Figure 4

$$
\begin{array}{c|cc|c|c}
 & z_2 & z_3 & & q_i = \\
\hline
 & -1 & \textcircled{1} & 9/4 = x_1 & 4x_1^4x_2 = 4(9/4)^4(4/3)^2 \\
 & 1 & -3 & (4/3)^3 = z_1 & 0 \\
 & 1 & -2 & (4/3)^2 = x_2 & x_1^4x_2 = (9/4)^4(4/3)^2 \\
\hline
r_j = & -3(9/4)^4(4/3)^2 & 2(9/4)^4(4/3)^2 & &
\end{array}
$$

Figure 5

$$
\begin{array}{c|cc|cc}
 & z_2 & x_1 & & q_i = \\
\hline
 & -1 & 1 & 9/4 = z_3 & 0 \\
 & -2 & 3 & (4/3)^3(9/4)^3 = z_1 & 0 \\
 & -1 & 2 & (4/3)^2(9/4)^2 = x_2 & x_1{}^4x_2 \\
\hline
r_j = & -x_1{}^4x_2 & 2x_1{}^4x_2-4x_1{}^4x_2 & &
\end{array}
$$

Figure 6



Solution for first objective function (1,9) occurs at intersection of $x_1 = 1$ and $x_1^2x_2 = 9$
Solution for second objective function (1.25,5.8) occurs along constraint $x_1^2x_2 = 9$
FEASIBLE REGION
x_2
x_1
9
8
7
6
5
4
3
2
1
1
2
3
4
5
6
7

Figure 7

The tableau of Figure 4 with dual variable calculations for the new objective function is shown in Figure 8. We see that we should try to increase z_3 as in the previous example. From Figure 8 we see that $x_1 = (9/4)z_3^{-1}$ and $x_2 = (4/3)^2 z_3^2$ so that the auxiliary problem is to:

$$\text{Minimize } g(z_3) = (9/4)^4(4/3)^2 z_3^{-2} + 2(4/9)(4/3)^2 z_3^3.$$

Differentiating g with respect to z_3, setting $g'(z_3) = 0$ and solving gives the solution:

$$x_1 = 1.25\ , \qquad x_2 = 5.80$$

$$z_1 = 13.97\ , \qquad z_2 = 1, \quad z_3 = 1.81$$

$$g(x_1, x_2) = 23.28$$

The final dual solution shows that this solution is optimal as shown in Figure 9. Figure 7 shows the graph of $g(x_1, x_2) = 23.28$ touching the boundary of the feasible set at the optimum point.

7. COMPUTATIONAL RESULTS

Some computational results are shown in Tables 1, 2 and 3. The program used was written in Basic and run on a Burroughs' 6700. Processing time included all central processing time except that used for input/output. The probem parameters were generated randomly. The exponents for the constraints were integers between minus one and three while the right hand sides were integer numbers between one and one hundred. The exponents in the objective function were random integers between zero and four multiplied by one half. Finally, the term coefficients in the objective function were random integers between zero and nine.

In the three tables a pivot means the same as in linear programming i.e., a basis change. A pass is the operation of modifying the value of some variable thereby changing the solution. In some cases, of course, this modification will lead to a pivot but not always. As we would expect from linear programming, pivots seem to be very fast. Problems with a high proportion of total passes being pivot operations have comparatively low solution times. As is obvious from Table 1 problems with many constraints relative to variables are solved mostly through pivoting. This fact seems to result from the reduced feasible region and the resulting high probability of finding the optimum at an extreme point or, at least, the intersection of several constraints. This can be verified again in Table 3. As can be seen in Table 2, more complex objective functions also seem to block out extreme point solutions.

In presenting these computational results certain considerations must be made concerning the program itself:

ONE DIMENSIONAL SEARCH PROCEDURE. Since pivots can be done rapidly compared to searching, the search procedure is not used unless pivoting would yeild a worse or equal solution than the initial value (with the exception that for degenerate situations pivoting is always performed). This criteria seems to reduce the total number of searches which must be performed thereby reducing total time. When searching in undertaken a combination quadratic fit and Fibonacci search

$$\begin{array}{c|cc|c|l}
 & z_2 & z_3 & & q_i = \\
\hline
 & -1 & 1 & 9/4 = x_1 & 4(9/4)^4(4/3)^2 - 2(9/4)^{-1}(4/3)^2 = 180.67 \\
 & 1 & -3 & (4/3)^3 = z_1 & 0 \\
 & 1 & -2 & (4/3)^2 = x_2 & (9/4)^4(4/3)^2 + 2(9/4)^{-1}(4/3)^2 = 47.14 \\
\hline
r_j = & -133.53 & 86.39 & &
\end{array}$$

Figure 8

$$\begin{array}{c|cc|c|l}
 & z_2 = 1 & z_3 = 1.81 & & q_i = \\
\hline
 & -1 & 1 & 1.25 = x_1 & 46.56 \\
 & 1 & -3 & 13.97 = z_1 & 0 \\
 & 1 & -2 & 5.80 = x_2 & 23.28 \\
\hline
r_j = & -23.28 & 0 & &
\end{array}$$

Figure 9

Constraints	Phase I		Phase II			Processing Time (Sec.)
	Variables	Pivots	Variables	Pivots	Passes	
5	15	6	10	0	1	1.1
5	20	5	15	4	6	2.5
5	30	5	25	4	58	48.0
10	25	11	15	1	1	1.3
10	30	14	20	5	6	3.5
10	35	16	25	8	29	24.3
15	45	21	30	12	13	7.5
15	50	26	35	16	21	17.5
15	55	25	40	24	44	47.4
20	45	25	25	1	1	4.2
20	60	37	40	6	6	10.5
25	55	30	30	3	3	7.4
25	60	42	35	3	3	10.7
25	65	47	40	6	6	13.8
25	70	54	45	15	15	20.5
25	75	45	50	27	46	95.5
30	70	52	40	5	5	16.8
30	80	73	50	22	22	37.5
30	85	75	55	23	23	39.2
35	100	68	65	22	22	51.6

Table 1: Time results. Objective function has five terms - no unconstrainted variables.

Terms in Objective Function	Phase I Pivots	Phase II		Processing Time (Sec.)
		Pivots	Passes	
5	5	4	6	2.5
10	5	3	5	3.9
15	5	3	35	73.7

Table 2: Time results. 5 constraints 20 variables in Phase I; 15 variables in Phase II - no unconstrainted variables.

Unconstrained Variables	Constraints	Variables		Phase I Pivots	Phase II		Processing Time (Sec.)
		Phase I	Phase II		Pivots	Passes	
0	5	15	10	6	0	1	2.5
5	5	15	10	5	0	9	8.7
0	10	30	20	14	5	6	3.5
10	10	30	20	10	1	21	28.3

Table 3: Time results. Objective function has five terms.

procedure is used. This portion of the program was written to take advantage of the speed of quadratic prediction and the robustness of the Fibonacci search. Search is stopped only when the machine tolerance (twelve significant digits) in the objective function is reached, i.e., the predicted points are so close that the objective functions are equal to within twelve significant digits.

STOPPING RULES. Stopping rules in nonlinear programming are by necessity extremely complex. Straight limits on the maximum allowable size of the reduced costs associated with movable variables is difficult since the importance of these reduced costs can only be considered in terms of the absolute size of the objective function. Also, when the one dimensional search is used the resultting minimum solution is only an approximation which leads to nonzero reduced costs associated with the last variable moved. (Exact solutions in the liner search would, of course, lead to a zero reduced cost for this variable). Typically even small deviations in the linear search seem to yield large nonzero reduced costs - thus the choice of as fine a tolerance as possible in the search portion of the program. From these two considerations two stopping rules were developed both of which lead to program termination. The first rule is used after a variable is modified but no pivot has occurred. The reduced cost associated with this variable should be zero but will never actually equal zero. During the next pass the reduced costs are tested against the reduced cost, say r_m, of the variable just modified. If no other variable which is a candidate for modification under section 3 of the algorithm has a reduced cost whose absolute value exceeds $2r_m$ then the program stops and the present solution is printed out as optimum. The second criteria is invoked after a pivot. This rule stems from the fact that the reduced cost is $v_j \partial g/\partial v_j$. We wish to stop when the objective function is changing by extremely small amounts relative to the objective function. In our program if no candidate for modification has a reduced cost whose absolute value exceeds 10^{-7} g(x) or (10^{-5} whichever is greater) then the program stops and the present value is printed out as the optimum. The stopping rules for the computational results displayed in Tables 1, 2 and 3 were purposely very stringent. Figure 10 shows the convergence properties of two of the longer runs. In these situations the objective function levels off very quickly. These results indicates that for practical purposes the times shown are quite high.

CODING EFFICIENCY. Use of equation (7) when calculating partial derivatives enhances the program considerably since for a given solution the bulk of the calculation $H \odot x$ stays constant across all variables. It should be pointed out that similar efficients were not coded into the search routine. In particular the objective function is reevaluated for each new point of the search procedure even though only m+1 variables of the total variables are being modified. The linear search itself seems to be time consuming which suggests that improvements here are possible. Indeed it may prove to be true that high precision in these searches is not necessary early in the solution procedure.

8. EXAMPLE PROBLEM

Due to exogenous economic factors an automobile manufacturer is faced with a large inventory of completed new automobiles. The company sells three basic types of cars - large, medium and small. Each size has a different number of cars in inventory. The manufacturer also knows that sales of one type car has a significant effect on sales of the other two types. The manufacturer wishes to determine a short term price structure which will allow him to maximize revenue while turning a significant proportion of his surplus inventory.

Sales of each type of car are forecasted via the following Cobb-Douglas equation:

sales of car type i = S_i =

$$K_i p_1^{\alpha_{i1}} p_2^{\alpha_{i2}} p_3^{\alpha_{i3}} \qquad i = 1,2,3$$

where p_j are the prices for the three cars.

Revenue from total sales is of course $\sum_i S_i P_i$. Assume we know the following parameters which predict sales:

$$\alpha_{ij} = \begin{array}{c|ccc} i \backslash j & 1 & 2 & 3 \\ \hline 1 & -.6 & .4 & .8 \\ 2 & .45 & -.5 & .4 \\ 3 & .5 & .3 & -.5 \end{array}$$

$$K_i = \begin{array}{|c|} \hline 1 \times 10^3 \\ \hline 8 \times 10^3 \\ \hline 3 \times 10^3 \\ \hline \end{array}$$

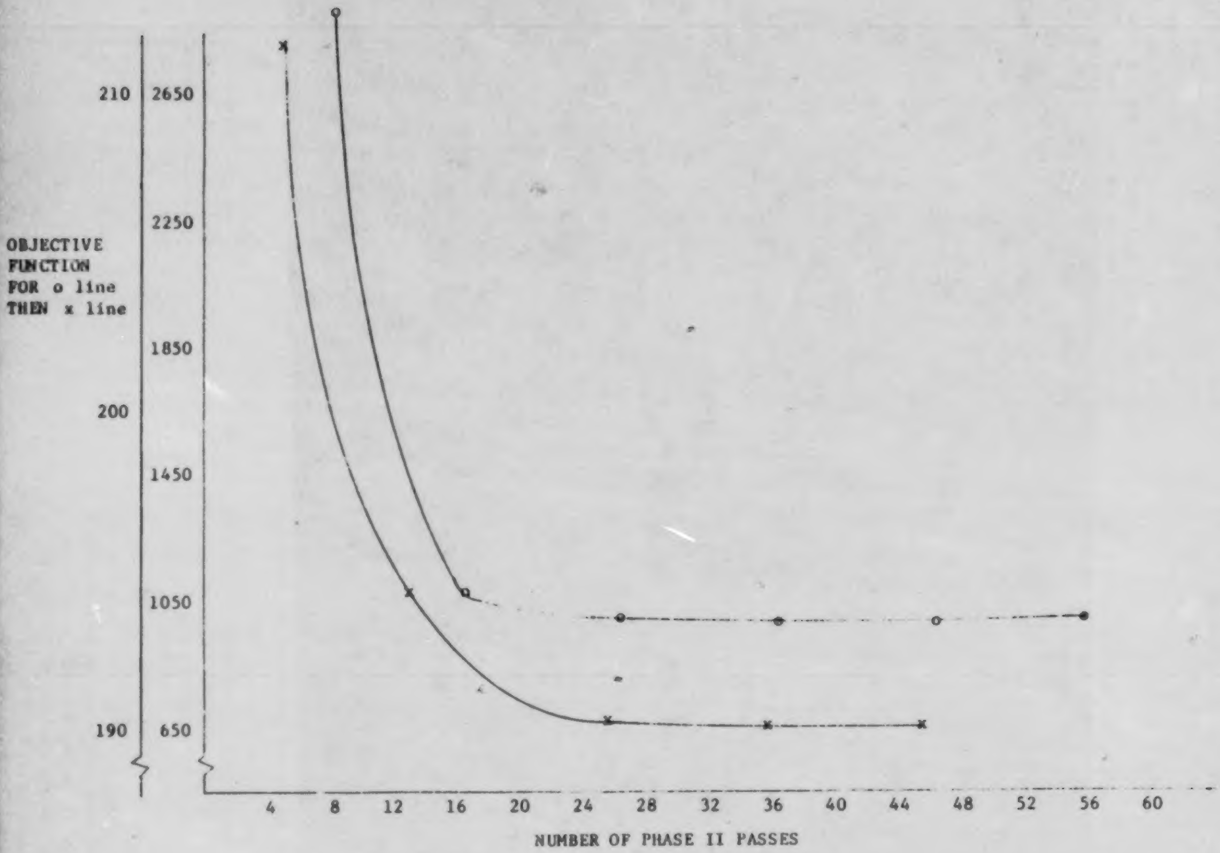


Figure 10: Convergence of the Algorithm

and

$$\text{Inventory} = I_1 = \begin{bmatrix} 45{,}000 \\ 128{,}000 \\ 150{,}000 \end{bmatrix}$$

$$\text{Minimum Desired Sales} = \begin{bmatrix} 25{,}000 \\ 64{,}000 \\ 90{,}000 \end{bmatrix}$$

Maximum and minimum prices as set by management are:

Lower bound	Upper bound
none	none
none	5,000
1,000	3,000

Management wishes to determine a price for each car which maximize net short run revenue from sales.

The monomial model for this problem will be:

$$\text{Max. } (1p_1^{.4}p_2^{.4}p_3^{.8} + 8p_1^{.45}p_2^{.5}p_3^{.4} + 3p_1^{.5}p_2^{.3}p_3^{.5}) \times 10^3$$

$$\text{s.t. } 25\text{x}10^3 \leq 1\text{x}10^3\ p_1^{-.6}p_2^{.4}p_3^{.8} \leq 45\text{x}10^3$$

$$64\text{x}10^3 \leq 8\text{x}10^3\ p_1^{.45}p_2^{-.5}p_3^{.4} \leq 128\text{x}10^3$$

$$90\text{x}10^3 \leq 3\text{x}10^3\ p_1^{.5}p_2^{.3}p_3^{-.5} \leq 150\text{x}10^3$$

$$0 \leq p_1$$

$$0 \leq p_2 \leq 5{,}000$$

$$1{,}000 \leq p_3 \leq 3{,}000$$

The algorithm described in this paper took 1.8 seconds to solve the above model. Prices and predicted sales are shown below

$$\text{Optimum Prices, } P_i = \begin{array}{c|c|} 1 & \$9030.75 \\ 2 & 5000.00 \\ 3 & 1526.94 \end{array}$$

$$\text{Sales for these prices} = \begin{bmatrix} 45{,}000 \\ 128{,}000 \\ 93{,}921 \end{bmatrix}$$

Net revenue generated from this short term pricing strategy will be $\$1.190\text{x}10^9$.

9. DISCUSSION

In this paper we have presented an efficient algorithm for the solution of monomial problems. Certain comments should be made here concerning the algorithm presented. (1) Although the program of the algorithm was written for polynomial objective functions this is not necessary. Any differentiable objective could be easily substituted for the one chosen in this paper. The one used however is totally general within the framework of polynomial functions. (2) The program was written to solve both constrained and unconstrained variable problems, it could easily be modified for upper bounded variables without the necessity of using them in the constraints. (3) As was stated earlier in the paper all the typical linear programming techniques can be easily modified to fit within the framework of monomial programs. (4) The algorithm itself is very similar to Zangwill's convex simplex method [9] and Wolfe's reduced gradient method [7] and [8]. However in our algorithm no transformation are required. (5) Computational results from this algorithm are very encouraging particularly in the light of possible computational improvements which can be made within the linear search routine.

BIBLIOGRAPHY

[1] Charnes, A., and M. Kirby, "Modular Design, Generalized Inverses and Convex Programming," Operations Research, Vol. 13, pp. 836-847, (1965).

[2] Gaver, D.P., and G.L. Thompson, Programming and Probability Models in Operations Research, Brooks/Cole, 1973.

[3] Shaftel, T.L., Y. Smeers, and G.L. Thompson, "A Simplex-Like Approach to a Class of Geometric Programming Problems," Management Sciences Research Report No. 282, GSIA, May 1972.

[4] Shaftel, T.L., Y. Smeers, and G.L. Thompson, "A Simplex-Like Approach for Nonlinear Programs with Nonlinear Constraints," Management Sciences Research Report No. 285, GSIA, July 1972.

[5] Shaftel, T.L., and G.L. Thompson, "Computational Experience with the Continuous Modular Design Problems," Management Sciences Research Report No. 309, March 1973.

[6] Shaftel, T.L., and G.L. Thompson, "The Continuous Multiple Modular Design Problem," Management Sciences Research Report No. 254, GSIA, January 1972.

[7] Wolfe, P., "Methods of Nonlinear Programming," in Recent Advances in Mathematical Programming, R. Graves and P. Wolfe (eds.), McGraw-Hill, New York, 1963, pp. 67-86.

[8] Wolfe, P., "Methods for Linear Constraints," in Nonlinear Programming, J. Abadie (ed.), North-Holland Publishing Co., Amsterdam, 1967, pp. 120-124.

[9] Zangwill, W., Nonlinear Programming: A Unified Approach, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1969.

IMPLEMENTATION AND USE OF NONLINEAR
COST MULTICOMMODITY FLOW SUBROUTINES

Hoang Hai Hoc
Department of Electrical Engineering
Ecole Polytechnique, Montreal, Canada

ABSTRACT

Several mathematical programming formulations of nonlinear cost multicommodity flow problems as well as a few of their applications and solution algorithms are considered. Three algorithms are implemented and tested on moderate size problems; they are adapted Frank and Wolfe's method, Darfemos' explicit path flow exchange procedure and specialized convex simplex algorithm. Experimental results seem to indicate that Frank and Wolfe's method is sufficiently competitive when some what less precise solutions are accepted. The specialized convex simplex algorithm is particularly suited to "post-optimization" tasks and is computationally superior to the explicit path flow exchange procedure.

1. THE PROBLEM AND ITS APPLICATIONS

We consider the following nonlinear cost multicommodity flow problem. Given a network with n nodes, let $\{1,2,\ldots,p\}$, $p \leq n$, denote the set of nodes which are either supply points (sources, origins) or demand points (sinks, destinations), or both. In addition, every node in the network can be used as transhipment point. Let A be the set of arcs in the network and D denote a p x p matrix whose (i, j) entry indicates the nonnegative number of units which must flow between nodes i and j. Let the cost of shipping x_{ij} units along (i, j) be $f_{ij}(x_{ij})$, where the cost is a function of the total flow along the arc. Let us use x^s_{ij} to denote flow along arc (i,j) with origin s, the multicommodity flow problem which we address is then

$$\text{Minimize} \quad f(X) = \sum_{(i,j)\in A} f_{ij}(x_{ij}) = \sum_{(i,j)\in A} f_{ij}\Big(\sum_{s=1}^{p} x^s_{ij}\Big) \qquad (1)$$

subject to

$$D(s,j) + \sum_{\substack{k=1 \\ (j,k)\in A}}^{n} x^s_{jk} = \sum_{\substack{i=1 \\ (i,j)\in A}}^{n} x^s_{ij} \qquad (2)$$

for all $j = 1, 2, \ldots, m$, $s = 1, 2, \ldots, p$, $j \neq s$

$$x^s_{ij} \geq 0 \text{ for all } (i,j)\in A,\ s = 1, 2, \ldots, p \qquad (3)$$

We remark that (1)-(3) represent what we call many (destinations) to one (origin) node-arc formulation of the multicommodity flow problem. A more intuitive form of the flow conversation equations (2) can be obtained by considering separately flow x^{st}_{ij} along arc (i,j) and associated with origin s and destination t. We have then the one (destination) to one (origin) node-arc formulation of the multicommodity flow problem.

Minimize

$$f(X) = \sum_{(i,j)\in A} f_{ij}(x_{ij}) = \sum_{(i,j)\in A} f_{ij}\Big(\sum_{s,t} x^{st}_{ij}\Big) \qquad (4)$$

subject to

$$\sum_{i \ni (i,j)\in A} x^{st}_{ij} - \sum_{k \ni (j,k)\in A} x^{st}_{jk} = \begin{cases} -D(s,t), & j = s,\ j = 1, 2, \ldots, n \\ 0, & j \neq s,t \quad s,t = 1, \ldots, p \\ D(s,t), & j = t,\ t \neq s \end{cases} \qquad (5)$$

$$x^{st}_{ij} \geq 0 \text{ for all } (i,j)\in A,\ t,s = 1,2,\ldots,p \quad t \neq s \qquad (6)$$

However, the most intuitive formulation of the problem considered is perhaps the arc-chain formulation. This formulation is derived by providing, for each (s,t) commodity, a set P_{st} of paths from node s to node t and path flows x^{st}_q for all $q \in P_{st}$.

Minimize

$$f(X) = \sum_{(i,j)\in A} f_{ij}\Big(\sum_{s,t,q} a^{stq}_{ij}\, x^{st}_q\Big) \qquad (7)$$

subject to

$$\sum_{q\in P_{st}} x^{st}_q = D(s,t) \text{ all } s,t = 1,2,\ldots,p,\ s \neq t \qquad (8)$$

$$x^{st}_q \geq 0 \text{ all } q\in P_{st},\ s,t = 1,2,\ldots,p,\ s \neq t \qquad (9)$$

where

$$a^{stq}_{ij} = \begin{cases} 1, & \text{if arc } (i,j) \text{ belongs to path } q \in P_{st} \\ 0, & \text{otherwise.} \end{cases} \qquad (10)$$

The nonlinear multicommodity network flow formulations presented above have been used as mathematical programming models for interesting and practical problems in transportation networks, in packet switched computer communications networks, etc. We call the reader's attention to two specific applications: the equilibrium traffic assignment problem in a road network [1,2,3,17,19,20] and the optimal static routing problem in a computer communications network [8,13]. A more general multicommodity flow problem based on the arc-chain formulation mentioned above and including continuous arc capacity augmentation can be found in Reference [14], where arc capacities are explicitly accounted for and the total cost is not necessarily a function separable or not, of only total flows on each arc.

2. THE SOLUTION ALGORITHMS

It is possible to present various algorithms for solving the nonlinear cost multicommodity flow problem considered within the general framework of either Zoutendijk's feasible direction method [19] or the Conditional Gradient Method [14]. Since our principal interest is testing specialized algorithms, we choose to restrict ourselves to the first alternative. Then, the feasible direction method for solving the problem of minimizing a continuously differentiable function f from R^{μ} into R over a convex polyhedral set $F = \{X \in R^{\mu} : G_{\lambda x \mu} X = 0, X \geq 0\}$ can be stated as follows.

Algorithm A-0

```
(1) Find an initial feasible solution $X^o \in F$.  Set
    $k = 0$.
(2) If $X^k$ does not satisfy the stopping criteria,
    then go to (3).
(3) Find a descent direction $d^k$ with respect to
    $X^k$, i.e. a vector $d^k \in R^{\mu}$ such that
        $(X^k + td^k) \in F$, for all $t \in [0, \sigma]$,
        $f(X^k + td^k) < f(X^k)$, for all $t \in (0, \gamma)$
(4) Find a solution $t^k$ to the one-dimension search
    problem
        Minimize $\{f(X^k + td^k) \mid t \in [o, \sigma]\}$
(5) Set $X^{k+1} = X^k + t^k d^k$, $k = k + 1$, and go to (2).
```

A close examination of the algorithm A-0 reveals that the determination of descent directions plays on important role in characterizing different specialized versions of the feasible direction method as applied to various problems. A simple way to obtain such a direction, which consists in linearizing the objective function, was suggested by Frank and Wolfe [7]. In fact, let y^k be an optimal solution of the following linear programming problem

$$\text{Minimize } \{f(X^k) + \nabla f(X^k)^T (y - X^k) \mid y \in F\} \qquad (11)$$

then,

$$d^k = y^k - X^k$$

defines a descent direction with respect to X^k, provided that X^k is not an optimal point.

The first specialized version of the feasible direction method for solving the nonlinear cost multicommodity flow problem can be obtained by observing that in this particular case the L.P. problem (11) is equivalent to computing shortest path flow with respect to the distances

$$C_{ij} = \left.\frac{df_{ij}}{dx_{ij}}\right|_{x_{ij} = x^k_{ij}} \quad \text{for all } (i,j) \in A$$

The specialized algorithm obtained [2,8,17] can be stated as follows.

Algorithm A-1

```
(1) Let the current feasible solution be
    $X^k = (x^k_{ij} \mid (i,j) \in A)$
(2) Compute $C_{ij} = \left.\frac{df_{ij}}{dx_{ij}}\right|_{x_{ij} = x^k_{ij}}$ and
    set $Y_{ij} = 0$ for all $(i,j) \in A$. Set $s = 1$.
(3) Find the shortest path between node s and
    every node in the network using the $C_{ij}$'s as
    distances.
(4) For each destination t in the network, set
    $y_{ij} = y_{ij} + D(s,t)$ for every arc $(i,j)$ on the
    shortest path between node s and node t.
(5) If s is not equal to the number of origins,
    set $s = s + 1$ and go to step 3; otherwise go
    to step 6.
(6) Let $y^k = (y_{ij} \mid (i,j) \in A)$. Minimize the function
    f along the line segment between $X^k$ and $y^k$,
    using a one-dimensional search technique.
    Let the minimizing point be $X^{k+1}$.
(7) Test the stopping criterion, and go to step 2
    if it fails; otherwise, stop.
```

Another way of obtaining a descent direction at a feasible solution X is described below. It is more involved than Frank and Wolfe's procedure, and very similar to the simplex algorithm for L.P. problems.

Let Q be a basis of the constraint coefficient matrix G. By partitionning the variables into basic and non-basic sets, indexed by I and J respectively, and renumbering them if necessary, one can choose as a feasible direction any direction

$$d = (d_I, d_J)$$

satisfying the following conditions:

$$d_I = -[Q^{-1}R]d_J$$

$$\begin{aligned} &d_j \geq 0 \quad \text{for all } j \in J \text{ such that } x_j = 0 \\ &d_i \geq 0 \quad \text{for all } i \in I \text{ such that } x_i = 0 \end{aligned} \qquad (12)$$

and

$$[\nabla f(X)_J - (Q^{-1}R)^T \nabla f(X)_I]^T d_J < 0 \qquad (13)$$

Choose

$$r_I = 0$$

$$r_J = \nabla f(X)_J - (Q^{-1}R)^T \nabla f(X)_I$$

$$r_{i_1} = \min_i \{r_i\}$$

$$r_{i_2} x_{i_2} = \max_i \{r_i x_i\}$$

one can obtain either a descent direction of reduced gradient type

$$\begin{aligned} &d_j = \begin{cases} -r_j, \text{ if } r_j \leq 0, \\ -r_j x_j, \text{ otherwise; } \quad \text{for all } j \in J \end{cases} \\ &d_I = -(Q^{-1}R)\, d_J \end{aligned} \qquad (14)$$

or a descent direction of convex simplex type

$$\begin{aligned} &d'_j = \begin{cases} 1 & \text{if } j = i_1 \\ 0 & \text{if } j \in J,\ j \neq i_1 \end{cases} \\ &d''_j = \begin{cases} -1 & \text{if } j = i_2 \\ 0 & \text{if } j \in J,\ j \neq i_2 \end{cases} \\ &d_J = \begin{cases} d'_J & \text{if } |r_{i_1}| > r_{i_2} x_{i_2} \\ d''_J & \text{otherwise} \end{cases} \\ &d_I = -(Q^{-1}R)\, d_J \end{aligned} \qquad (15)$$

under the following assumption:

(a) $X_I > 0$ (the basis Q is non-degenerate)

or

(b) for every $i \in I$, $\frac{\delta f}{\delta x_i} \leq \frac{\delta f}{\delta x_j}$ for all $j \in J$.

In the following algorithm we will restrict ourselves to the reduced gradient descent direction and the arc-chain formulation since the convex simplex descent direction is a special case of the reduced gradient descent direction. This algorithm solves a series of restricted problems defined over a series of restricted sets of allowed paths. It uses a sequential optimization approach obtained by decomposing a restricted problem into a series of subproblems, each corresponds to an origin-destination (O-D) pair. Successive restricted problems are obtained by the path (column) generation technique described below [16,19].

Algorithm A-2

(1) Let be given for each O-D pair (s,t) associated with a positive D(s,t) a restricted set of allowed paths $P^0_{s,t}$ from node s to node t, and an arbitrary path flow $\{h^{st}_q \mid q \in P^0_{s,t}\}$. Set $\ell = 0$.

(2) Solve the restricted multicommodity flow problem over the restricted set of allowed paths $P^\ell = \bigcup_{(s,t)} P^\ell_{s,t}$ as follows.

(2.1) Set $i_T = 1$, $i_{opt} = 0$.

(2.2) If i_{opt} is equal to the total number of O-D pairs M, go to (3). Otherwise, set $i_T = i_T + 1$, $k = i_T$ modulo $(M) + 1$, $\ell = 1$.

(2.3) Solve the subproblem associated with the k^{th} O-D pair (s,t) as follows.

(2.3.a) Compute

$$c_{ij} = \frac{df_{ij}(x_{ij})}{dx_{ij}} \text{ for all } (i,j) \in A,$$

$$u_q = \sum_{(i,j)\in A} a^{stq}_{ij}\, c_{ij} \text{ for all } q \in P^\ell_{s,t},$$

$$u_b = \min_{q \in P_{st}} \{u_q\}.$$

(2.3.b) Compute

$$d_q = \begin{cases} (u_b - u_q)\, h^{st}_q, \text{ if } h_q > \epsilon_1 \text{ and } \frac{|u_b - u_q|}{u_b} > \epsilon_2, \\ 0, \text{ otherwise.} \end{cases}$$

$$d_b = \sum_{\substack{q \in P^\ell_{s,t} \\ q \neq b}} (-d_q)$$

(2.3.c) If $d_b > 0$, then go to (2.3.d). Otherwise, the flows on paths in P_{st} satisfy the optimality conditions. Then, if $\ell \leq 1$ let $i_{opt} = i_{opt} + 1$ and go to (2.2); otherwise, i.e. $\ell \geq 2$, let $i_{opt} = 1$ and go to (2.2).

(2.3.d) Search for t* minimizing

$$\sum_{(i,j)\in A} f_{ij}(x_{ij} + tw_{ij})$$

where

$$w_{ij} = \sum_{q \in P_{st}} a^{stq}_{ij} d_q$$

and

$$0 \le t \le \min\{-\frac{h^{st}_q}{d_q} \mid q \in P_{st}, d_q < 0\}$$

(2.3.e) Let $h^{st}_q = h^{st}_q + t^* d_q$, all $q \in P_{st}$,

$$x_{ij} = x_{ij} + t^* w_{ij}, \text{ all } (i,j) \in A,$$

$$\ell = \ell + 1, \text{ and go to (2.3.a).}$$

(3) For every O-D pair (s,t), find a shortest path b_{st} from s to t with respect to the distances

$$c_{ij} = \frac{df_{ij}(x_{ij})}{dx_{ij}}, \text{ for all } (i,j) \in A.$$

Let u_{st} be the length of b_{st}, and z_{st} be the maximum length for all significantly positive flow paths in P^ℓ_{st}.

Set

$$P^{\ell+1}_{st} = \begin{cases} P^\ell_{st} \cup \{b_{st}\}, & \text{if } \dfrac{z_{st} - u_{st}}{z_{st}} > \epsilon_2, \\ P^\ell_{st}, & \text{otherwise.} \end{cases}$$

$$P^{\ell+1} = \bigcup_{s,t} P^{\ell+1}_{st}$$

If $P^{\ell+1} \subseteq P^\ell$ then stop; otherwise, set $\ell = \ell + 1$ and go to (2).

Close examination of Algorithm A-2 reveals that it would be a cumbersome, if not clumsy, bookkeeping task to store all positive flow paths, to check and to update all path flows, separately. Storage requirements could be fantastically large, even for mass storage devices, if the networks considered are of considerable size. Fortunately, there exists an equivalent, but very compact representation of path flows based on the node-arc formulation of multicommodity flow problems. Moreover, time-consuming shortest path calculations can be avoided completely by using a generalized version of the augmented precessor index method [9] to solve the flow subproblem associated with an origin node k, while maintaining all other flows constant. These considerations have given rise to the following algorithm [20].

Algorithm A-3

(1) For every origin node k, find arbitrary flow $(x^k_{ij}, (i,j) \in A)$ and spanning arborescence E(k) rooted at k. Let $i_T = -1$ and $i_{opt} = 0$.

(2) If i_{opt} is equal to the number of origins N_S, then an optimal solution is attained. Otherwise, let $i_T = i_T + 1$ and $k = i_T$ modulo $(N_S) + 1$.

(3) Solve the flow problem associated with the origin node k as follows.
(a) Let $\ell = 0$ and compute the distances

$$c_{ij} = \frac{df_{ij}(x_{ij})}{dx_{ij}} \quad \text{for all } (i,j) \in A.$$

(b) Compute for each node $i \ne k$ the length of the path J_{ki} from k to i in E(k): $z_i = \sum_{(u,v) \in J_{ki}} c_{uv}$.

Compute for each arc $(i,j) \notin E(k)$ the reduced length

$$r_{ij} = c_{ij} + z_i - z_j$$

(c) Define

$$I_1 = \{(i,j) \notin E(k) \mid \frac{r_{ij}}{z_i} < -\epsilon_2\}$$

$$I_2 = \{(i,j) \notin E(k) \mid \frac{r_{ij}}{z_i} > \epsilon_2 \text{ and } x^k_{ij} > \epsilon_1\}$$

If either I_1 or I_2 is nonempty go to (d). Otherwise; the flows x^k_{ij} originating at k satisfy the optimality conditions. Thus, if $\ell \le 0$ then let $i_{opt} = i_{opt} + 1$ and go to (2); Otherwise, let $i_{opt} = 1$ and go to (2).

(d) Determine (i_1, j_1) and (i_2, j_2) such that

$$r_{i_1 j_1} = \min_{I_1}[r_{uv}] \text{ and } r_{i_2 j_2} x^k_{i_2 j_2} = \max_{I_2}[r_{uv} x^k_{uv}].$$

If $|r_{i_1 j_1}| \ge r_{i_2 j_2} x^k_{i_2 j_2}$, then let $(\bar{i}, \bar{j}) = (i_1, j_1)$ and $\alpha = +1$; Otherwise let $(\bar{i}, \bar{j}) = (i_2, j_2)$ and $\alpha = -1.0$.

Retrace the cycle

$$L_{\bar{i}\bar{j}} = (J_{k\bar{i}} \cup \{(\bar{i},\bar{j})\} \cup J_{k\bar{j}} = L^+_{\bar{i}\bar{j}} \cup L^-_{\bar{i}\bar{j}}$$

Let

$$d_{uv} = \begin{cases} 0 & (u,v) \notin L_{\bar{i}\bar{j}} \\ \alpha & (u,v) \in L^+_{\bar{i}\bar{j}} \\ -\alpha & (u,v) \in L^-_{\bar{i}\bar{j}} \end{cases} \text{ for all } (u,v) \in A.$$

(e) Determine

$$\sigma = x^k_{i_3 j_3} = \begin{cases} \min\{x^k_{uv} \mid (u,v) \in L^-_{\bar{i}\bar{j}}\} & \text{if } \alpha = 1 \\ \min\{x^k_{uv} \mid (u,v) \in L^+_{\bar{i}\bar{j}}\} & \text{if } \alpha = -1 \end{cases}$$

If $\sigma > \epsilon_1$, go to (g); otherwise, perform the following pivoting operation.

(f) Let $E(k) = E(k) \cup \{(u,v)\} - \{(i_3, j_3)\}$ where (u,v) is an arbitrary element of the set

$$I_3 = \{(u,v) \notin E(k) \mid v = j_3 \text{ and } x^k_{uj_3} > x^k_{i_3 j_3}\}$$

(If I_3 is empty replace (i_3, j_3) by its sucessor on $L^+_{\bar{i}\bar{j}}$ or $L^-_{\bar{i}\bar{j}}$ and determine again a new I_3, and so on) and go to (b).

(g) Find t^* minimizing

$$\sum_{(i,j) \in L^-_{\bar{i}\bar{j}}} f_{ij}(x_{ij} + td_{ij}) \text{ for } t \in [0, \sigma]$$

(h) Let

$$x^k_{uv} = x^k_{uv} + t^* d_{uv},$$

$$x_{uv} = x_{uv} + t^* d_{uv},$$

$$c_{uv} = f_{uv}(x_{uv}), \text{ for all } (u,v) \in L^-_{\bar{i}\bar{j}}$$

Set $\ell = \ell + 1$ and go to (b).

3. ALGORITHM IMPLEMENTATION, TESTING AND USE

In this section we discuss the implementation of a basic version for each of the algorithms A1, A2, and A3 presented in Section 2. Basic versions of the algorithms were programmed and tested on relatively moderate size problems. The tests were planned to gain insight into the computational behavior of the algorithms.

(3.a) For test problems on networks with about a hundred nodes and a thousand arcs, the implementation of algorithm A1 does not present any serious problem. At iteration k of the algorithm it is necessary to find the shortest paths for all O-D pairs with respect to the distances equal to the marginal costs of traveling on arcs at current flows X^k. Shortest path flows Y^k are obtained by sending for every O-D pair (s,t) the flow requirement D(s,t) along the shortest path joining s and t. Floyd's algorithm [6] has been used to determine the shortest paths between all pairs of nodes of a network. This choice has been made in order to facilitate programming tasks. Computing time could be significantly reduced by repeatly using Dijkstra's algorithm [4], especially when the number of origins N_S is much smaller than the total number of nodes. The Golden Section method [15] has been used for searching a minimum of the function f along the line segment $[X^k, Y^k]$. The algorithm is started from the initial shortest path flows with respect to any estimated distances.

Two test networks are considered: one is the 76 arc, 24 node network of the City of Sioux Falls, South Dakota [17], and the other is the 376 arc, 155 node network of the City of Hull, Canada [19]. Global computational results are presented in Table 1 for comparison with the algorithms A2 and A3. We remark that for the basic version implemented, computing time per iteration of Algorithm A1 is independent of the number of O-D pairs. Consequently, for the Sioux Falls' network, we solved only one test problem with 528 O-D pairs. Moreover to reveal the convergence speed of algorithm A1, we have plotted in Figure 1 the value of the objective function, maximum percentage change in flow components, and computing time as functions of the number of iterations.

A convergence difficulty inherent to Frank and Wolfe's algorithm occurs when the objective function assumes a minimum point in the interior of the feasible region. In such a case, the direction indicated by the gradient may change very rapidly as the sequence of points approaches the optimal solution. Consequently, the points may zig zag around the optimal solution. In our test problems, although the gradient of the objective function can not be zero at the optimal solution (since the objective function is strictly increasing and has no stationary point in the feasible region) the convergence seems to slow down appreciably when approaching the optimal solution.

(3.b) The basic version of algorithm A2 is essentially the computer program given in [16], implementing Dafermos' algorithm using a column generation technique. In this variant of algorithm A2 the descent direction in (2.3 b) is actually replaced by the following

$$u_a = \min (u_q \mid q \in P^\ell_{st})$$

$$u_b = \max (u_q \mid q \in P^\ell_{st}, h^{st}_q > 0)$$

$$d_a = 1$$

$$d_b = -1$$

$$d_q = 0 \text{ for all } q \in P^\ell_{st}, q \neq a, q \neq b$$

An appropriate move along d means shifting flow from the maximum cost path to the minimum cost path in order to equalize travel costs on these paths.

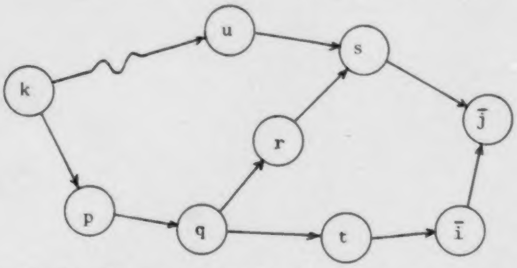
The program has been modified in order to accomodate much more O-D pairs than the maximum number of 20 originally allowed. These minor modifications are straight-forward and consist in reducing the core storage required to memorize path and path flow data. This reduction is easily achieved by performing flow adjustments for one O-D pair at a time. Only data associated with the O-D pair considered need to be in core. When all path flows for this O-D pair satisfy the optimality conditions, updated data are written on disks for latter uses.

(3.c) The essential part of algorithm A3 consists in solving an one-commodity flow problem associated with an origin node. This is achieved by a generalized version of the augmented predecessor index method. For strongly connected networks, it is also possible to work only with a basis which can be represented by a spanning arborescence $E(k)$ rooted at origin node k. For the arborescence representation, we use triple-label notation [12] in which each node i receives three labels

- child (i) pointing to first node j such that $(i,j) \in E(k)$,
- parent (i) pointing to the node j for which $(j,i) \in E(k)$, and
- sibly (i) pointing to node $j \neq i$ such that $(\text{parent } (i), j) \in E(k)$.

Node numbers z_i are then readily obtained by traversing arborescence $E(k)$ in end-order. Reduced arc lengths computed from node numbers and marginal costs permit us to verify the optimality conditions, and to choose an out-of-kilter arc for which some flow adjustment is performed.

Let $(\bar{i}, \bar{j})$ be the out-of-kilter arc considered, and let us suppose that the reduced length $r_{\bar{i}\bar{j}}$ is negative. Thus, flow adjustment consists in transfering flow from $(q, r, s, \bar{j})$ to $(q, t, \bar{i})$ to reduce $r_{\bar{i}\bar{j}}$ to zero. The same adjustment process as that used in [16], i.e. an interval halving method, is adopted, since in this process no iterations are required if integral (lump) flow transfer occurs. This speeds up the computation, expecially at the beginning of the solution process, and gives better results than the golden section method.



Basis changes are frequently required. To perform a basis change, a zero flow arc (r,s) in $E(k)$ is replaced by a positive flow arc (u,s). Such an arc (u,s) exists necessarily, if flow on (s,j) is positive. Otherwise, $(s,\bar{j})$ is considered instead of (r,s), and can in fact be replaced by $(\bar{i},\bar{j})$. After a basis change, if arc (u,v) is inserted into the new arborescence, then only sub-arborescence rooted at u needs to be considered for purpose of updating node numbers.

To start the algorithm, initial arborescences rooted at different origin nodes as well as arbitrary initial flows are required. We calculate the shortest path from an origin to every node in the network by means of Dijkstra's algorithm. At the end of the shortest path calculation we obtain the shortest path flows, and for each node the shortest distance as well as its predecessor node on the shortest path from the origin to the node itself. This does mean that we have for each origin an arborescence represented in predecessor index notation, which is easily transformed into triple-label notation and used for basis representation. The shortest path flows obtained are used as initial flows.

(3.d) We will now discuss experimental results obtained from solution of the test problems by means of algorithms A-1, A-2, and A-3.

First, it is clear from the statement of the algorithms and presentation of the basic versions implemented of that algorithms A-2 and A-3 are quite similar. For both algorithms we may interpret the underlying itterative computational process within the framework of traffic equilibrium as a process by which rational drivers acquire knowledge about their travel cost, switch to cheaper routes, congesting them, and causing a new flow pattern to envolve. This interpretation gives rise to a realistic stopping criterion for algorithms A-2 and A-3, which requires that the most expensive and the cheapest used paths as well as the cheapest unused path from an origin to a destination must not differ more than, say, 5% in travel cost. Such stopping criteria may also be used to generate solutions simulating a well accepted fact that α% of drivers do not care for differences of β% in travel cost between paths from their origin to their destination.

Second, although flow adjustments in algorithms A2 and A3 are quite similar, algorithm A3 uses a more compact representation of flows without explicitly storing all paths allowed. Thus, we may expect algorithm A3 to provide better performances than algorithm A2 in terms of core and mass storage requirements as well as computing time since less book-keeping is necessary and the shortest path calculation avoided. The experimental results obtained confirm this conjecture in a very striking manner. Figure 2 shows a relation between computing time and number of O-D pairs for a series of 4 Sioux Falls network test problems presented in Table 1. The computing time of algorithm A2 is seven times greater than that of A3. All runs were executed on an IBM/System 360- Model 50 with Ampex ECS under OS-MVT, using Fortran IV Level G Compiler.

Third, the computational behavior of algorithm A1 is significantly different from that of algorithms A2 and A3. Defining an iteration as a shortest path flow calculation (essentially an execution of Floyd's algorithm) followed by an one-dimensional search, the value of the objective function, maximum percentage change in flow components, and computing time are shown iteration by iteration in Figure 1. The results indicate that improvement per iteration is very substantial at the beginning of the iterative process, but that the rate of convergence slows when approaching the optimal solution. This means that increasing the precision of the solutions is a time consuming task.

Such a characteristic could be ennoying for people attempting to use algorithm A1 in selecting network topology or network improvement projects.

(3.e) We are in fact searching for an efficient network configuration evaluation procedure to be used in selecting network improvement projects subject to a budget constraint. For a regional road network, where congestion may reasonably be negligible, we have proved a superadditive property of a natural objective function and devised a powerful branch search procedure [10], which was later greatly improved and reported in [5]. Interesting results presented in [5] also show that heuristics, related to those suggested in [10], give solutions extremely close to the optimal ones. We are presently developping similar procedures for networks where congestion is no longer negligible [11] In such a case it is necessary to perform post-optimality analysis by removing a link from or adding a link to a network configuration. Algorithm A3 seems particularly suited to this reoptimization task. Table 2 shows computing time required as function of link insertions/deletions.

REFERENCES

[1] M.J. Beckman, C.B. McGuire, and C.B. Winsten, "Studies in the economics of transportation", Yale University Press, New Haven, Conn., 1956.

[2] M. Bruynooghe, A. Gilbert, and M. Sakarovitch, "Une méthode d'affectation du trafic", in W. Lentzback and P. Baron (Eds), Strassenbahn und Strassenverkehrstechnik, Proceedings of the 4th International Symposium on the Theory of Traffic Flow, pp. 198-204, Bonn, 1969.

[3] S.C. Dafermos and F.T. Sparrow, "The traffic assignment problem for a general network", J. Res. Nat. Bureau of Standards-B, Vol. 73B, pp. 91-118, 1969.

[4] E.W. Dijkstra, "A note on two problems in connexion with graphs", Numerische Mathematik, Vol. 1, pp. 269-271, 1959.

[5] R. Dionne, "Recherche d'un algorithme efficace pour le problème du choix optimal d'un réseau", Publication No. 174, Département d'informatique, Université de Montréal, Mai 1974.

[6] R.W. Floyd, "Algorithm 97: Shortest Path", Communications of the ACM, Vol. 5, p. 345, 1962.

[7] M. Frank and P. Wolfe, "An algorithm of quadratic programming", Naval Research Logistics Quarterly, Vol. 3, pp. 95-110, 1956.

[8] L. Fratta, M. Gerla, and L. Kleinrock, "The flow deviation method: An approach to store-and-forward communication network design", Networks, Vol. 3, pp. 97-133, 1973.

[9] F. Glover, D. Karney, and D. Klingman, "The augmented predecessor index method for locating stepping stone paths and assigning dual prices in distribution problems", Transportation Science, Vol. 6, pp. 171-180, 1972.

[10] Hoang Hai Hoc, "A computational approach to the selection of an optimal network", Management Science, Vol. 19, pp. 488-498, 1973.

[11] Hoang Hai Hoc, "Network Improvements via Mathematical Programming", Paper presented at the IX International Symposium on Mathematical Programming, Budapest, August 23-27, 1976.

[12] E. Johnson, "Networks and Basic Solutions", Operations Research, Vol. 14, pp. 619-623, 1966.

[13] L. Kleinrock, "Analytic and Simulation Methods for Computer Network Design", 1970 SJCC, AFIPS Conf. Proc., Vol. 36, pp. 569-579, AFIPS Press, Montvale, N.J. 1970.

[14] R. W. Klessig, "An Algorithm for Nonlinear Multicommodity Flow Problems", Networks, Vol. 4, pp. 343-355, 1974.

[15] J. Kowalik and M.R. Osborne, "Methods of Unconstrained Optimization problems", Elsevier, N.Y., 1968.

[16] T.L. Leventhal, G.L. Nemhauser, and L.E. Trotter, "Traffic assignment: Computer program II", DT-FHA Report No. 25, Department of Operations Research, Cornell University, November 1971.

[17] L.J. Leblanc, E.M. Morlok, and W.P. Pierskalla, "An efficient approach to solving the road network equilibrium traffic assignment problem", Transportation Research, Vol. 9, pp. 309-318, 1975.

[18] E.F. Moore, "The shortest path through a maze", Proc. of the Int. Symp. on the Theory of Switching, pp. 285-292, Harvard University Press, Cambridge, Mass., 1959.

[19] S. Nguyen, "Une approche unifiée des méthodes d'équilibre pour l'affectation du trafic", Publication No. 171, Département d'informatique, Université de Montréal, Mars 1974.

[20] S. Nguyen, "An algorithm for the traffic assignment problem", Transportation Science, Vol. 8, pp. 203-216, 1974.

	Number of O-D pairs	A-1	A-2	A-3
Sioux Falls Network (24 nodes, 76 arcs)	167	* * *	(541, 2.831)	(72, 2.821)
	252	* * *	(675, 215.9)	(98, 217.0)
	376	* * *	(783, 107.3)	(112, 107.3)
	528	(133, 76.55)	(1020,73.42)	(145, 72.79)
Hull Network (155 nodes, 376 arcs)	720	(1615, 4.023)	* * *	(283, 3.981)

Table 1 - Computational results for the algorithms A-1, A-2, and A-3: (Computing time in seconds, normalized final value of the objective function).

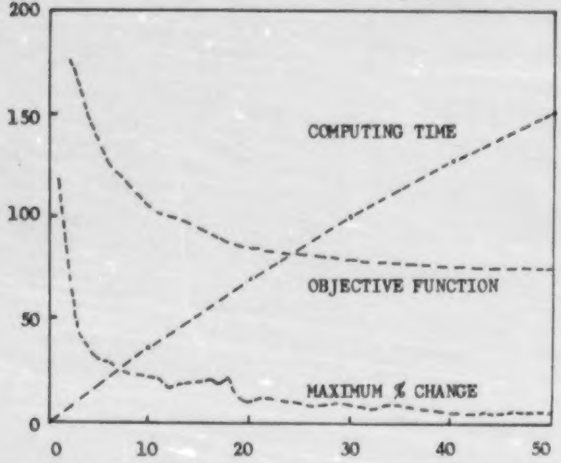


Figure 1.a.- Computational indices vs iteration number (Sioux Falls Network, Algorithm A1).

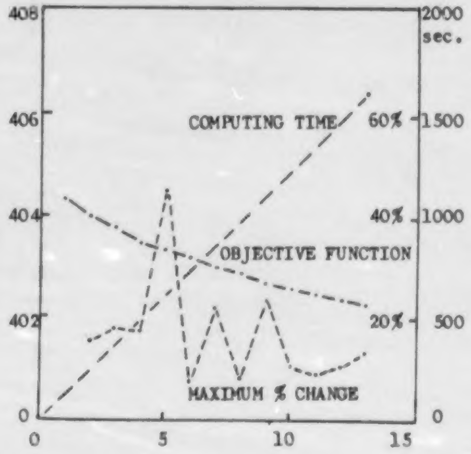


Figure 1.b.- Computational indices vs iteration number (Hull Network, Algorithm A1).

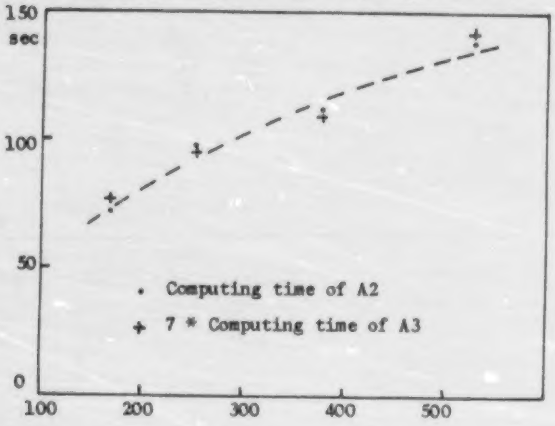


Figure 2- Computing time vs number of O-D pairs.

Number of successive link modifications	Average computing time required in seconds
0	145
1	154
2	186
3	218
4	266
5	269

Table 2 - Network reoptimization after link modifications.

IMPLICIT REPRESENTATION OF TRIANGULARITY CONSTRAINTS IN LINEAR PROGRAMS

Gamini Gunawardane
University of Sri Lanka and
University of Chicago

Linus Schrage
University of Chicago

ABSTRACT

In certain linear programs, for example, those arising out of production-inventory type problems, a large number of constraints have a special form. Examples are: simple upper bounds, generalized upper bounds, variable upper bounds, and generalized variable upper bounds. This paper identifies a class of constraints which enables us to obtain a large triangular submatrix in any feasible basis. Also presented is a method for representing these constraints implicitly within a modified version of the revised simplex method.

1. Introduction

From the early fifties on, there has been a steady progression of optimization strategies and tactics for solving large linear programs.[1] Such large LP's are common in problems in production and operation management. Examples are those derived from mixed integer linear programs arising out of such problems as production-inventory planning and facilities location. Consideration of multi-periods and/or multi-products increases the size of these problems substantially. Another factor contributing to the increase in size in some problems is the possibility of tighter formulations of integer programs as opposed to loose formulations. Importance of such tighter formulations is well explained by Schrage (1975b).

One area of research on large LP's is that of tactical variations in pricing and pivot selection. Another area, which is the area of our interest, is the exploitation of special structures to reduce storage and computation time by avoiding explicit consideration of a full inverse. Examples of such special structure constraints which have been represented implicitly are simple upper bounds (SUB), generalized upper bounds (GUB), and more recently variable upper bounds (VUB) (cf. Schrage (1975a)), and generalized variable upper bounds (GVUB), cf., Schrage (1975b). Mention should also be made of angular structures and factorizations described in Lasdon (1970) and Graves and MacBride (1973).

This paper identifies a class of constraints appearing in certain dynamic production management problems which allow us, by judicious row and column permutation, to obtain a large lower triangular submatrix in any feasible basis of the problem. Section 2 introduces these constraints and illustrates with examples instances where such constraints occur. Section 3 discusses the importance of having a large lower triangular submatrix. Section 4 outlines the detailed development of a solution method based on a modified version of the revised simplex method. Implementation considerations currently underway are also discussed.

2. Triangularity Constraints

In this section we define triangularity constraints and also give some examples of where they arise. First define

> Eligible set of row i = the variables - including slacks - that have a strictly positive coefficient on the LHS of row i; i.e. $[x_j : a_{ij} > 0]$.

Now we define triangularity constraints.

> A set of constraints of a linear program is called triangularity constraints - - or T constraints - - if
>
> (a) their RHS are nonnegative, and
> (b) the set of constraints can be permuted so that in a feasible basis to the problem, a variable in the eligible set of any T constraint does not have a nonzero element in any other T constraint lying above this constraint.

Example 1: One Machine, Multiproduct, Multiperiod Scheduling Problem with Setups.

$$\text{Min} \sum_{p=1}^{P}\sum_{t=1}^{T} k_p z_{tp} + \sum_{p=1}^{P}\sum_{i=1}^{T-1}\sum_{j=i+1}^{T} (j - i)h_p r_p x_{ijp} \qquad (1)$$

---

[1] For a recent review of some of this research see Beale (1975) and also the preface of the same publication.

$$\text{s.t.} \quad \sum_{i=1}^{j} x_{ijp} \cdot r_p = d_{pj} \qquad \text{all } p, j \qquad (2)$$

$$\sum_{p=1}^{P} y_{ip} \leq 1 \qquad \text{all } i \qquad (3)$$

$$z_{ip} \geq y_{ip} - y_{i-1,p} \qquad \text{all } i, p \qquad (4)\text{T}$$

$$(\text{or, } \; y_{ip} - y_{i-1,p} - z_{ip} \leq 0)$$

$$x_{ijp} \leq y_{ip} \qquad \text{all } i, j, p \qquad (5)\text{T}$$

$$x_{ijp} \geq 0 \qquad \text{all } i, j, p \qquad (6)$$

$$y_{ip}, z_{ip} = \{0, 1\} \qquad \text{all } i, p \qquad (7)$$

where

d_{pt} = demand for product p in period t

r_p = units of p which can be produced in period t

k_p = setup cost for product p

h_p = one period holding cost for product p

x_{ijp} = fraction of production capacity in period i used to satisfy product p demand in period j

$$y_{ip} = \begin{cases} 1 & \text{if product } p \text{ is produced in period } i \\ 0 & \text{else} \end{cases}$$

$$z_{ip} = \begin{cases} 1 & \text{if a setup is made in period } i \text{ for product } p \\ 0 & \text{else.} \end{cases}$$

Note that the LP relevant to our discussion is the relaxation: Min (1), s.t. (2) - (6). The letter T against constraints (4) and (5) indicates that they are T constraints. Now consider these two sets of constraints together.

For any RHS constant which is zero, without loss we can assume it to be a small positive number. Thus one of the variables with a positive coefficient in the LHS of each constraint of (4) and (5) must be in a feasible basis. For these variables of (5), there are no other nonzero elements in their columns within (4) and (5). This is no surprise as (5) is in fact of VUB form. In (4), the y_{ip} columns have no elements above the positive element corresponding to the row in which a particular y_{ip} is basic in a row of (4). Thus (4) and (5) qualify to be T constraints.

The immediate advantage in recognizing the T constraints is that we are able to get any feasible basis into the form:

$$\left[\begin{array}{c|c} C & D \\ \hline & \\ E & T \end{array}\right] \qquad (8)$$

where T is composed of constraints (4) and (5), and is lower triangular. Note that in order to get this form we had to place (4) and (5) in that order at the bottom of the constraints.

Example 2: Dynamic Plant Location Problem (Roodman and Schwarz, 1974)

$$\text{Min} \quad \sum_{i,t} F_{it} y_{it} + \sum_{i,j,t} c_{ijt} x_{ijt} \qquad (9)$$

$$\text{s.t.} \quad \sum_{i} x_{ijt} = 1 \qquad \text{all } j, t \qquad (10)$$

$$-y_{it} + y_{i,t-1} \leq 0 \qquad \text{all } i, t \qquad (11)\text{T}$$

$$x_{ijt} \leq y_{it} \qquad \text{all } i, j, t \qquad (12)\text{T}$$

$$x_{ijt}, y_{it} \geq 0 \qquad (13)$$

$$y_{it} \text{ is integer } (0, 1) \qquad (14)$$

where

F_{it} = fixed cost of operating plant i in period t

c_{ijt} = cost of transporting a unit from i to j in period t

x_{ijt} = fraction of requirement at j supplied by i in period t

$$y_{it} = \begin{cases} 1 & \text{if plant } i \text{ is "open" in period } t \\ 0 & \text{else.} \end{cases}$$

The LP relaxation is: Min (9), s.t. (10) - (13). Now, following arguments similar to those in example 1, we can get any feasible basis into form (8). (11) and (12) become the triangularity constraints.

To summarize, the motivation behind identifying T constraints was the ability to get any feasible basis into the form (8). The T constraints form the lower triangular T, and in the examples given above it would be seen that T is fairly large. Thus, considerable savings could be anticipated if these T constraints are implicitly represented. To do this, we exploit some special properties of triangular matrices.

3. Usefulness of the Triangular Submatrix

The following properties of triangular matrices make it convenient to have a large lower triangular submatrix in the basis.

(i) In our problem T is nonsingular. Thus T^{-1} exists. T^{-1} is also lower triangular.

(ii) T^{-1} can be computed by iterative substitution. (In general, T^{-1} is not explicitly needed.)

(iii) Pre-multiplication of matrices or vectors by T^{-1} can be done without finding T^{-1} explicitly. For instance, let R be an $n \times m$ matrix and we want $F = T^{-1}R$. If we partition F and R by columns,

$$f_i = T^{-1} r_i \ .$$

Now if we solve the system $Tf_i = r_i$ ($i = 1, 2, \ldots, n$), by back substitution we can get $T^{-1}R$ in $mn^2/2$ multiplications. In using the revised simplex method pre-multiplications of this type occur several times.

(iv) Both (ii) and (iii) are simplified if T has further special structures, e.g., band structures. Some practical problems tend to have such band matrices in place of T.

4. Implicit Representation of T Constraints

We saw earlier that for problems with T constraints a feasible basis could be written in the form:

$$B = \left[\begin{array}{c:c} C & D \\ \hdashline E & T \end{array}\right]$$

With this form of B, we have

$$B^{-1} = \begin{bmatrix} (C - DT^{-1}E)^{-1} & (C - DT^{-1}E)^{-1}(-D)T^{-1} \\ T^{-1}(-E)(C-DT^{-1}E)^{-1} & T^{-1}(-E)(C-DT^{-1}E)^{-1}(-D)T^{-1}+T^{-1} \end{bmatrix} . \qquad (15)$$

We plan to carry only the smaller inverse $(C - DT^{-1}E)^{-1}$ and not the larger B^{-1}. Information in the coefficient matrix A would be carried in part explicitly and in part using pointers and link lists. The next step is to apply the revised simplex method to this reduced working basis. To do this, we write the entire LP in standard form:

$$\text{Max } z = - \sum_{j=1}^{N} a_{0j} x_j$$

$$\text{s.t.} \quad \sum_{j=1}^{N} a_{ij} x_j = a_{i0} \quad i = 1, 2, \ldots, M$$

$$x_j \geq 0 \ .$$

Constraint $z + \sum_{j=1}^{N} a_{0j} x_j = 0$ will be the top row (row zero) in the tableau. If we had the full B^{-1}, the top row of this would have been the vector of dual prices $\{\pi_i\}$. Also, let m ($< M$) be the last explicit row, i.e., T constraints are in rows $m + 1, \ldots, M$.

Construction of the Pricing Vector

All elements of the pricing vector $\{\pi_i\}$ can be computed from the smaller inverse and the other information we carry. If $i \leq m$, then π_i is simply element i of row zero of $(C - DT^{-1}E)^{-1}$. For $i = m$, we can compute π_i starting from $i = M$ as follows. Denote by $b(i)$ the column basic in row i. Then,

$$\text{for } \ i = M: \quad \sum_{k=0}^{m} \pi_k a_{kj} + \pi_M a_{Mb(M)} = 0$$

$$\text{i.e., } \pi_M = - \left[\sum_{k=0}^{m} \pi_k a_{kj} \right] \Big/ a_{mb(M)}$$

$$\text{for } \ i = M - 1: \quad \sum_{k=0}^{m} \pi_k a_{kb(M-1)} + \pi_{m-1} a_{m-1,b(M-1)}$$

$$+ \pi_M a_{Mb(M-1)} = 0 \ .$$

In general for $m < i \leq M$

$$\sum_{k=0}^{m} \pi_k a_{kj} + \sum_{p=i}^{M} \pi_p a_{pb(p)} = 0 \ .$$

In fact, we do not always have to start computing π_i's from the last row. Certain π_i's, namely those corresponding to columns of T which have only the diagonal element nonzero, are also directly computed. This property is useful in implementing the pricing procedure. When we need a π_i, $i > m$, but do not have the price of a later row, we find the column basic in that row and price it out. This loop will produce the required π_i as soon as we reach a column with only the diagonal element nonzero. In a typical implementation we would price out only a small portion of the columns at any iteration. If none of the columns priced have elements in row i, then that π_i is not computed. The fact that a_{ij} are usually simple elements (0, 1, -1) also helps in reducing multiplications.

Selecting an Entering Column

This is done by computing $\sum_{k=0}^{M} \pi_k a_{kj}$ for selected nonbasic columns. The only saving here possibly would be by using the knowledge that some of the a_{kj}'s are +1 or -1.

Generating the Updated Representation of the Entering Column

Define the following notation:

$a_{\cdot j}$ = the column vector consisting of $a_{0j}, a_{1j}, \ldots, a_{Mj}$

$\alpha_{\cdot j} = B^{-1}a_{\cdot j}$

α_{ij} = element i of $\alpha_{\cdot j}$, $i = 0, 1, \ldots, M$.

The notation $(i;k)$ when used as a subscript denotes elements i through k along the associated dimension, e.g., $a_{\cdot j} = a_{(0;M)j}$.

If j is the entering column, then we want to get $B^{-1}a_{\cdot j} = \alpha_{\cdot j}$. From the previously encountered form of B^{-1}, this gives:

$$\alpha_{(0;m)j} = (C - DT^{-1}E)^{-1}(I: \; -DT^{-1})a_{\cdot j} \tag{16}$$

$$\alpha_{(m+1;M)j} = [(T^{-1}(-E)(C - DT^{-1}E): \; T^{-1}(-E)$$
$$(C - DT^{-1}E)^{-1}(-D)T^{-1} + T^{-1}]a_{\cdot j} \tag{17}$$

Consider (16): simplifying further,

$$\alpha_{(0;m)j} = (C - DT^{-1}E)^{-1}(a_{(0;m)j} + (-DT^{-1})$$
$$a_{(m+1;M)j})$$

$(DT^{-1})a_{(m+1;M)j}$ is computed easily if we use the special properties of T stated in section 2.

Let

$$f = T^{-1}a_{(m+1;M)j}.$$

Then

$$Tf = a_{(m+1;M)j}$$

and we solve this system as in Section 2. Usually $a_{(m+1;M)j}$ contains only a few nonzero elements, and we start with the top-most nonzero element in $a_{(m+1;M)j}$. The row corresponding to it gives the corresponding f_i. The next step is to compute Df. In an implementation currently under way, we plan to store D by columns. f is quite sparse and we need access only those columns of D corresponding to the few nonzeros of f. Because $T^{-1}a_{(m+1;M)j}$ is a column vector, $DT^{-1}a_{(m+1;M)j}$ is easily computed. Recalling that $(C - DT^{-1}E)^{-1}$ is the smaller inverse we are carrying, $\alpha_{(0;m)j}$ is now available.

Consider (17) simplifying further,

$$\alpha_{(m+1;M)j} = T^{-1}(-E)[(C - DT^{-1}E)^{-1}: \; (C - DT^{-1}E)^{-1}$$
$$-(D)T^{-1}]a_{\cdot j} + T^{-1}a_{(m+1;M)j}$$
$$= T^{-1}(-E)\alpha_{(0;m)j} + T^{-1}a_{(m+1;M)j}$$
$$= T^{-1}[a_{(m+1;M)j} - E\alpha_{(0;m)j}]$$
$$= f - T^{-1}E\alpha_{(0;m)j} \; .$$

We already have f and $\alpha_{(0;m)j}$. Now we compute $E \cdot \alpha_{(0;m)j}$, and pre-multiply by T^{-1}. Pre-multiplication by T^{-1} follows the simpler method already discussed.

$\alpha_{(m+1;M)j}$, and hence the entire $\alpha_{\cdot j}$, is now available.

Selection of the Pivot Row

This is done very much in the usual fashion by computing ratios. We already have the updated version of the entering column, and we need the updated RHS:

$$\alpha_{\cdot 0} = B^{-1}a_{\cdot 0} \; .$$

$\alpha_{(0;m)0}$ is given by:

$$\alpha_{(0;m)0} = (C - DT^{-1}E)^{-1}(\alpha_{(0;m),0} - DT^{-1}\alpha_{(m+1;M)0})$$

We could follow the same steps in the previous section and compute $\alpha_{(0;m)0}$, but a better way to carry $\alpha_{(0;m),0}$ explicitly and update each pivot by considering it as another column of $(C - DT^{-1})E^{-1}$.

$\alpha_{(m+1;M)0}$ is computed by back substituting the values of variables basic in implicit rows that have nonzero coefficients in a given explicit row. We need only the elements of $\alpha_{(m+1;M)0}$ which correspond to $\alpha_{(m+1;M)j} \neq 0$. The actual procedure to get the required α_{i0}'s by back substitution is similar to the looping discussed in constructing the pricing vector where the staircase structure of T is exploited.

Pivoting

The type of pivot to be performed depends on whether the departing column is basic in an explicit row or an implicit row. Let us consider the several cases that may arise.

Let

c = entering column

d = departing column

r = row in which column d is basic.

Case (i). Departing column basic in an explicit row.

Column c replaces d in $(C, E)'$. The change affects only C and E, and the effect is to to change only one column in $(C - DT^{-1}E)$. The new column in $(C - DT^{-1}E) = a_{(0;m)c} - DT^{-1}a_{(m+1;M)}c$. In computing the updated version of c, we had

$$\alpha_{(0;m)c} = (C - DT^{-1}E)^{-1}(a_{(0;m)c} - DT^{-1}a_{(m+1;M)c}).$$

Thus the new $(C - DT^{-1}E)^{-1}$ is obtained by using $\alpha_{(0;m)C}$ as pivot column and pivoting on element α_{rc}. Thus, this pivot is identical to a conventional revised simplex pivot.

Case (ii) a. Departing column basic in an implicit row, and c can directly replace d in row r, i.e., $r > m$ and $a_{ic} = 0$ for $m < i < r$, $a_{rc} = 1$.

Now a column is changed in $(DT)'$. The difficult part in this case, as well as in the next case, (ii) b, is identifying which columns of $(C - DT^{-1}E)$ have been affected by this column change.

Let 0/n as a subscript denote the old/new matrix; e.g., T_0^{-1} is T^{-1} before change of a column in $(DT)'$.

Let

$$R = (C - DT_0^{-1}E)^{-1} .$$

We have to repivot in any row i $(i = 1, \ldots, m)$ for which

$$R(a_{(0;m)b(i)} - DT_n^{-1}\, a_{(m+1;M)b(i)}) \neq e_i ,$$

where e_i is a 0 vector except for a 1 in row i.

The change of a column in $(D, T)'$ does not affect $a_{(0;m)b(i)}$. It affects $DT^{-1}a_{(m+1;M)b(i)}$ if column $b(i)$ has a nonzero in E in any row that communicates with row r of T. In other words, there will be no effect if $a_{(q;r)b(i)} = 0$, where q is the largest numbered row in T above r, which has only a lone nonzero element (the +1 in the diagonal) and such that all $a_{rk} = 0$ for $k < q$. The reason for this lies in the fact that under these circumstances, the solution to

$$T \cdot f = a_{(m+1;M)b(i)}$$

remains the same even after the change in T. Once we have identified a column $b(i)$ for which all zeroes in $a_{(q;r)b(i)}$ are not zero, we compute

$$a_{(0;m)b(i)} - D_n T_n^{-1}\, a_{(m+1;M)b(i)}$$

using the new D and T. Premultiplication of this by the current R gives us the pivot column. The pivot is made in row i.

Case (ii) b. Departing column basic in an implicit row, but c cannot replace d in row r, i.e., the entering column not of special form as in case (ii) a.
This means that the entering column (call it t) cannot replace d. If it did, it would destroy the partial triangular structure. As in GVUB, we handle this in two steps.

Step 1. Find a column c currently basic in an explicit row that can replace d. Such a column exists, because the new basis--with t in place of d--is a feasible basis, and it too should be adjustable to form (8).

Once c is found, step 2 is like case (ii) a. As many pivots as there are columns changed in $(C - DT^{-1}E)$ would be needed to update its inverse.

Step 2. Now replace c by t.

This is like case (i), and requires a pivot on a_{pt} using $\alpha_{(0;m)t}$ as the pivot column.

Though it is more complicated than the VUB and GVUB implicit representation, the implicit representation of triangularity constraints also saves much computation in the early steps of the revised simplex method. The pivoting step, particularly case (ii) a, may involve several pivot operations, but this may well be compensated by the savings due to having these operations performed in the smaller inverse instead of the full inverse. Storage-wise, implicit representation is efficient in that it enables us to store problems with a large number of triangularity constraints entirely in main memory.

REFERENCES

Beale, E. M. L. "The Current Algorithmic Scope of Mathematical Programming," Mathematical Programming Study 4 (December 1975).

Graves, W. G. and R. D. McBride, "The Factorization Approach to Large-Scale Linear Programming," Working Paper, Graduate School of Management, UCLA (August 1973).

Hadley, G. Linear Programming. Addison-Wesley, 1962.

Lasdon, L. S. Optimization Theory for Large Scale Systems. Macmillan, 1970.

Roodman, G. and L. Schwarz, "The Dynamic Plant Location Problem," Working Paper, University of Rochester (1974).

Schrage, L. S. "Implicit Representation of Variable Upper Bounds in Linear Programs," Mathematical Programming Study 4 (December 1975a).

________. "Implicit Representation of Generalized Variable Upper Bounds in Linear Programs," Report #7543, Center for Mathematical Studies in Business and Economics, Graduate School of Business, University of Chicago (1975b).

AN EFFICIENT GENERAL ALGORITHM FOR THE COMPUTATION OF LINEAR DECISION RULES

SIDNEY F. THOMAS

Caribbean Industrial Research Institute,
Tunapuna Post Office,
Trinidad.

ABSTRACT

The Z-transform technique is applied to the optimization problem consisting of a quadratic objective function and linear difference equations. A rigorous proof of the general solution is offered, and a computational method is described which is likely to be more efficient than matrix inversion or matrix iteration approaches.

INTRODUCTION

In a recent paper, Hay and Holt /4/ have presented a general solution for the optimization problem in which the objective function is quadratic, and the variables are subject to linear difference equations, utilizing a technique known as the Z-transform. This technique is an alternative to other solution methods currently in use, such as matrix inversion (e.g. Theil /7/ and matrix iteration (e.g. Bellman /1,p. 78 et.seq./, Chow /2/). While the developments in /4/ have contributed some additional insight into the class of quadratic-linear optimization problems, the Z-transform technique as presented seemed to have little computational significance, for as the authors stated, "... matrix iteration or matrix inversion are likely to be computationally superior approaches..". /op.cit. p. 257/.

In this paper, I return to the Z-transform technique for solving quadratic-linear optimization problems. I pay particular attention to the question of computational efficiency, and reach the conclusion that the Z-transform technique can in fact lead to solution methods computationally *superior* to either matrix iteration or matrix inversion, at least under certain stationarity assumptions, contrary to what is suggested in /4/. In arriving at this conclusion, I suggest an algorithm that is different from that in /4/, in the way certain characteristic roots are determined. These developments were made as part of an investigation /8/ of an educational planning problem, where because of the large number of system variables, and the long lead times involved (which result in difference equations of high order), computational efficiency is placed at a premium.

Altogether apart from the issue of computational efficiency, the proof of the general solution is different in certain respects from that found in /4/. Specifically, the proof is both simpler and more rigorous. Not only that, but the particular notational devices used in constructing the proof, lead naturally to the more efficient algorithm developed, in contrast to /4/, where the notational devices are such that it is not so easy to see the (computationally) important equivalence between the characteristic root problem, and the eigenvalue problem for a suitably constructed matrix. Thus this paper sets out to make a contribution at two levels: firstly, in respect of computational efficiency, and secondly in respect of a further consolidation of the theory underlying the use of Z-transform techniques for the quadratic-linear optimization problem.

This paper is structured into two parts. Part 1 describes the basic approach of the Z-transform method of solution, following /4/ but utilizing different notational devices. Part 2 is where the real contribution of this paper lies, addressing the two critical issues arising from the Z-transform approach, viz. (i) the question of the existence of certain roots needed in the solution procedure; and (ii) the question of an efficient computational routine for locating those roots. The relative efficiencies of alternative computational approaches are then compared, and the paper concluded.

PART 1 - THE Z-TRANSFORM APPROACH

In this part of the paper, the Z-transform approach to quadratic-linear optimization problems is described. Following a general statement of the problem, the (first-order) conditions for a maximum are derived. Some restricting (stationarity) assumptions are then introduced, and the Z-transform technique used to show how a solution could be derived from the system of linear equations defining the maximum conditions.

THE GENERAL PROBLEM

The problem under consideration, following Theil's /7/ canonical form, is the following:-

Find y which maximizes

$$\Phi\,(x,y) = a'x + b'y + \tfrac{1}{2}x'Ax + \tfrac{1}{2}y'By + \tfrac{1}{2}x'Cy + \tfrac{1}{2}y'C'x \tag{1a}$$

$$\text{s.t.} \quad x = Ry + s. \tag{1b}$$

That is, we have a quadratic criterion in two sets of variables x and y. The set of variables y are the *decision* variables, which (it is assumed) can be manipulated by the decision-maker. The set of variables x are called *state* variables. These are affected by the decision variables in accordance with the linear side relations given in equation (1b). The state variables are also affected by the set of variables s, which are uncontrollable from the decision-maker's viewpoint, and exogenously specified. The parameters of the problem are the vectors a and b, and the matrices A, B, C, and R.

The sets of variables x, y and s are time-partitioned as follows:-

$$\begin{aligned} x &= \left(\underline{x}_1', \underline{x}_2', \ldots, \underline{x}_T'\right)' \quad ST\times 1 \\ y &= \left(\underline{y}_0', \underline{y}_1', \ldots, \underline{y}_{T-1}'\right)' \quad DT\times 1 \qquad (2) \\ s &= \left(\underline{s}_1', \underline{s}_2', \ldots, \underline{s}_T'\right)' \quad ST\times 1 \end{aligned}$$

That is, for each of T periods in the planning horizon, there are S state variables, S exogenous variables, and D decision variables, so that x is of dimension ST×1, y is of dimension DT×1, and s is of dimension ST×1.

The vector and matrix parameters a,b,A,B,C,R, are of dimensions which conform to x and y. That is, a is ST×1, b is DT×1, A is ST×1, B is DT×DT, C is ST×DT and R is ST×DT.

The linear decision rule consists in finding the optimal *first*-period decisions, that is $\underline{y}_0^*$, when the planning horizon $T \to \infty$, as a function of the exogenous variables, the asterisk in $\underline{y}_0^*$, being used to denote optimality. Note here that when x, y and s are as defined in (2), the exogenous variable $\underline{s}_1$ can be expressed as a function of the most recently observed state $\underline{x}_0$. For this reason, writing a first-period linear decision rule as a function of the exogenous variables $\underline{s}_1, \underline{s}_2, \ldots,$ is equivalent to a feedback control law, (Chow /2/) in which the optimal decision in the t-th period, $\underline{y}_t^*$, is written as a function of the most recently observed state $\underline{x}_{t-1}$. In this paper, therefore, and contrary to Chow's /2, p. 17/ viewpoint, we take the practical view that we are dealing essentially with differences in computational method when we compare matrix iteration, matrix inversion and Z-transform methods for solving the problem stated.

MAXIMUM CONDITIONS

The formal first order conditions for a maximum, are obtained by substituting for x in (1a) by making use of (1b), and then differentiating and setting the result equal to zero (Theil /7/), as follows: Substituting (1b) into (1a), we get:-

$$\begin{aligned} \Phi(y) = a'(Ry + s) + b'y + \tfrac{1}{2}(Ry + s)'A(Ry + s) \\ + \tfrac{1}{2}y'By + \tfrac{1}{2}(Ry + s)'Cy + \tfrac{1}{2}y'C'(Ry + s) \end{aligned} \qquad (3)$$

i.e. $\Phi(y) = k_0 + k'y + y'Ky$ (4)

where,

$$k_0 = a's + \tfrac{1}{2}s'As \qquad (5a)$$

$$k = a'R + b' + R'As + s'C \qquad (5b)$$

$$K = \tfrac{1}{2}(R'AR + B + R'C + C'R) \qquad (5c)$$

Differentiating (4) with respect to y and setting the result equal to zero, we get the formal first order conditions for a maximum, thus:-

$$Ky^* = -k \qquad (6)$$

We are concerned in this paper only with methods for solving the system of equations given by the first-order conditions. The second-order conditions for a maximum are not discussed here, although it may be remarked in passing that sufficient, but not necessary, second-order conditions are that K be negative-definite. See Theil /7/ for a discussion of this issue.

ASSUMPTIONS

We now introduce the following assumptions:-

(i) Assume that A, B and C are real, band block-diagonal matrices, with (time -) invariant band, and that A and B are also symmetric.

(ii) Assume that R is lower band block-diagonal and real, with (time -) invariant band.

(iii) Assume that all variables and parameters are real, if not already so specified.

(iv) Assume that the parameters A, B, C and R have values such that the second-order conditions are satisfied.

Underlying assumptions (i) and (ii) is the more basic assumption that cross-temporal effects, both in the criterion function and in the system dynamics (the linear side-relations), are limited in the number of periods they span. Furthermore, the assumption is made that these cross-temporal effects are time-invariant, so that for any given cross-temporal span, the relevant parameters are the same, regardless of where in the planning horizon the span is actually taken. Considering the fact that we are letting $T \to \infty$, this would seem to be a necessary assumption in most cases.

The symmetry assumption for A and B can be made without loss of generality since we are dealing with quadratic forms. The assumption that R is *lower* band block-diagonal amounts to saying that current values of the state variables are dependent only on previous decisions, and not at all on future decisions, which would seem to be a necessary assumption in most problem situations.

The assumption that all parameters are real needs no justification. It would be difficult to give a practical interpretation of complex parameters in most applications.

Following on assumptions (i) and (ii), we have the result that K must necessarily be band block-diagonal, symmetric and real. This result can be quite easily proved, though with some tedium using the defining relation (5c).

Taking the relevant cross-temporal "span" of the system to be 'd' periods, therefore (this may be a system lead time; for example, in an educational system, the lead time between initial enrolment and final graduation may be considered to be (at most) six years, or periods, in which case d = 6), and with the symbol (') denoting transposition, we may write K as follows:-

$$K = \left[\begin{array}{cccccccccc}
K_d & K_{d-1} & \cdots & K_0 & & & & & & \\
K'_{d-1} & K_d & K_{d-1} & \cdots & K_0 & & & & & \\
\vdots & \ddots & & \ddots & & \ddots & & & & \\
K'_0 & \cdots & K'_{d-1} & K_d & K_{d-1} & \cdots & K_0 & & & \\
& K'_0 & \cdots & K'_{d-1} & K_d & K_{d-1} & \cdots & K_0 & & \\
& & \ddots & & \ddots & & \ddots & & \ddots & \\
& & & K'_0 & \cdots & K'_{d-1} & K_d & K_{d-1} & \cdots & K_0
\end{array}\right] \quad DT \times DT \qquad (7)$$

This is merely a general characterisation of a band block-diagonal, real, symmetric matrix. When $T \to \infty$, the infinite-band portion has "width" 2d+1, made up of the 2d+1 block elements $K'_0, \ldots, K'_{d-1}, K_d, K_{d-1}, \ldots, K_0$. Note that in this characterisation we must have that K_d is symmetric, which it is, from (5c). Note also that each block element has dimension D×D, and can be obtained from (5c) by first time-partitioning A, B, C, and R, into block elements conforming to Assumptions (i) and (ii), and then carrying out the indicated operations. The result (7) which then follows will form the basis for the method of solution to which we are headed.

METHOD OF SOLUTION

Substituting (7) into (6), letting $T \to \infty$, and taking the Z-transform[1] of the result, we get:

$$\sum_{i=0}^{2d-1} Q_i(z)\underline{y}_i^* + z^d F(z) Z(\underline{y}_t^*) = Z(\underline{k}_t) \qquad (8)$$

where Z(.) denotes the Z-transform of a (vector, in this case) sequence of variables; $Q_i(z)$, $i = 0, \ldots, 2d-1$ are the polynomial matrices that premultiply $\underline{y}_0^*, \underline{y}_1^*, \ldots, \underline{y}_{2d-1}^*$, which are obtained upon taking the Z-transform, and F(z) is the *characteristic polynomial (matrix)* associated with the infinite-band portion of (7). F(z) is defined by the following:-

$$F(z) = \sum_{i=0}^{d-1} z^i K'_i + \sum_{i=0}^{d} z^{d+i} K_{d-i} \qquad (9)$$

Note that for any arbitrary value of the variable z (8) is an equation in the (2d+1)D variables $\underline{y}_0^*, \underline{y}_1^*, \ldots, \underline{y}_{2d-1}^*$ and $Z(\underline{y}_t^*)$.

Now multiplying (8) through by $\hat{F}(z)$, the adjoint of F(z), and making use of the fact that for any general square matrix A, $\hat{A}A = |A|I$, where $\hat{A}$ denotes the adjoint of A, $|A|$ denotes the determinant of A, and I the identity matrix, we get:-

$$\hat{F}\left(\sum_{i=0}^{2d-1} Q_i \underline{y}_i^*\right) + z^d |F| Z(\underline{y}_t^*) = \hat{F}\, Z(\underline{k}_t) \qquad (10)$$

where the z arguments denoting the dependence of the polynomial matrices $F, \hat{F}, Q_0, Q_1, \ldots, Q_{2d-1}$ on z, have been dropped for notational simplicity.

Now, if we choose z within the unit circle such that $|F(z)| = 0$, the term in $Z(\underline{y}_t^*)$ drops out; and if further, the sequence $\{\underline{k}_t,\ t = 0,1,2,\ldots\}$ is bounded above, then $Z(\underline{k}_t)$ is finite.[2] We would thus have an equation in only the 2dD variables $\underline{y}_0^*, \underline{y}_1^*, \ldots, \underline{y}_{2d-1}^*$. Such a z would be by definition a root of the characteristic polynomial $|F(z)|$. Furthermore, we know from the result in Linear Algebra /3, p. 61/ that $\hat{F}(z)$ has rank unity for any root of $|F(z)|$. Consequently, substitution in (10) of any z_m, say, within the unit circle, which is a root of $|F(z)|$, yields exactly one linearly independent equation in the 2dD variables $\underline{y}_0^*, \underline{y}_1^*, \ldots, \underline{y}_{2d-1}^*$, obtained by taking a single non-zero row from $\hat{F}(z_m)$. Denoting such a row by $\hat{f}_m$(1×D), we have on substitution of a root z_m into (10), one linearly independent equation defined thus:-

$$\hat{f}_m\left(\sum_{i=0}^{2d-1} Q_i(z_m)\underline{y}_i^*\right) = \hat{f}_m Z(\underline{k}_t) \qquad (11)$$

In order to solve for the 2dD variables $\underline{y}_0^*, \underline{y}_1^*, \ldots, y_{2d-1}^*$, we need exactly 2dD linearly independent equations. We may obtain dD linearly independent equations from the original conditions for a local maximum. Observe there that the first dD equations contain only the 2dD variables $\underline{y}_0^*, \underline{y}_1^*, \ldots, \underline{y}_{2d-1}^*$. The remaining dD linearly independent equations may be obtained from equation (11), by substituting dD roots z_m satisfying $|z_m|<1$, $m = 1,2,\ldots,dD$. It is obvious that this procedure can only work if there are *exactly* dD roots of $|F(z)|$ lying within the unit circle of the complex plane. Furthermore, the computational efficiency of this procedure depends critically upon the efficiency with which the roots z_m, $m = 1,2,\ldots,dD$ can be found. The rest of this paper addresses itself to these two questions.

Note in passing that the occurence of complex roots poses no insuperable problems. See /4. 5, 8/ for methods of handling complex roots.

PART 2 - A PROOF OF EXISTENCE, AND A METHOD FOR FINDING THE ROOTS

In this part of the paper, firstly we prove the existence of exactly dD roots of $|F(z)|$ lying within the unit circle of the complex plane, using a different and more rigorous approach than Hay and Holt /4/. Secondly, we describe an efficient method for obtaining the dD roots of $|F(z)|$ lying within the unit circle. Thirdly, a brief comparison is made of the relative efficiencies of the alternative approaches.

EXISTENCE PROOF

To prove the existence of dD roots of $|F(z)|$ lying within the unit circle of the complex plane, the approach is as follows: The concept of a *symmetrical* polynomial (Definition 1) is introduced, and it is proved that symmetrical polynomials have an even number of roots, half of which lie within the unit circle of the complex plane (Theorem 1). It is then shown that $|F(z)|$ is a symmetrical polynomial of degree 2dD (Theorem 2), from which the required result immediately follows (Theorem 3). To prove that $|F(z)|$ is symmetrical, it is necessary to introduce the concept of the *reverse* of a polynomial (Definition 2) and to prove first, several lemnata involving this, and the concept of symmetrical polynomials.

DEFINITION 1: A *symmetrical*[3] polynomial is a polynomial of even degree 2n, say, $\sum_{r=0}^{2n} a_r x^r$ such that $a_r = a_{2n-r}$, $r = 0, \ldots, n$. Example:-

$$f(x) = 3 + 2x + 4x^2 + 2x^3 + 3x^4$$

is symmetrical.

DEFINITION 2: The *reverse* of an n-degree polynomial $\sum_{r=0}^{n} a_r x^r$ is defined to be the n-degree polynomial $\sum_{r=0}^{n} a_{n-r} x^r$. Let R{.} denote the reverse of a polynomial. Example:-

$$R\{3 + 2x + 4x^2\} = 4 + 2x + 3x^2$$

The reverse of an n-degree polynomial f(x) may be defined alternatively as follows:-

$$R\{f(x)\} \equiv x^n f(\tfrac{1}{x})$$

The equivalence of the two alternative definitions may be easily proved. Example:-

$$R\{3 + 2x + 4x^2\} = x^2(3 + \frac{2}{x} + \frac{4}{x^2}) = 4 + 2x + 3x^2$$

as before.

LEMMA 1: *If x is a root of the symmetrical polynomial $f(x)$, then its reciprocal $\frac{1}{x}$ is also a root. That is, if $f(x)$ is symmetrical, $f(x) = 0 \iff f(\frac{1}{x}) = 0$.*

Proof: The proof follows easily from the definition of a symmetrical polynomial, and is therefore omitted.

THEOREM 1: *A symmetrical polynomial of degree $2n$ possesses exactly n roots, x_i, $i = 1, \ldots, n$, some possibly repeated, satisfying $|x_i|<1$, $i = 1, \ldots, n$; provided only that there is no root x such that $|x| = 1$.*

Proof: For any (in general complex) number x, either $|x|<1$ or $|x|>1$, if $|x|\neq 1$.

Furthermore, $|x|>1 \iff |\frac{1}{x}|<1$.

Now, by Lemma 1, $f(x) = 0 \iff f(\frac{1}{x}) = 0$. Moreover, being of degree 2n, f(x) must have exactly 2n roots (some possibly repeated).

By symmetry, therefore, f(x) must have exactly n roots satisfying $|x|<1$ and exactly n satisfying $|x|>1$ (some possibly repeated); provided only that there is no root x such that $|x| = 1$. QED.

LEMMA 2: *The sum of symmetrical polynomials of the same degree is also symmetrical.*

Proof: The proof is obvious and therefore omitted.

LEMMA 3: *The sum of an even-degree polynomial and its reverse is a symmetrical polynomial. That is, $f(x) + R\{f(x)\}$ is symmetrical if $f(x)$ is of even degree.*

Proof: The proof follows easily from the definitions of a symmetrical polynomial and the reverse of a polynomial, and is therefore omitted.

LEMMA 4: *The reverse of a product of polynomials is the product of reverses of the individual polynomials making up the product. That is, $R\{f_1 \cdot f_2 \cdot \ldots \cdot f_p\} = R\{f_1\} \cdot R\{f_2\} \cdot \ldots \cdot R\{f_p\}$.*

Proof: The proof is by induction. It is shown first of all that if the result holds true for the product of p ($p \geq 2$) polynomials, that it must also hold for the product of p+1 polynomials.

Assume

$$R\{f_1 \cdot f_2 \cdot \ldots \cdot f_p\} = R\{f_1\} \cdot R\{f_2\} \cdot \ldots \cdot R\{f_p\}. \tag{12}$$

Then,

$$\begin{aligned} R\{f_1 \cdot f_2 \cdot \ldots \cdot f_p \cdot f_{p+1}\} &= R\{f_1 \cdot f_2\} \cdot R\{f_3\} \cdot \ldots \cdot R\{f_{p+1}\} \\ &= R\{f_1\} \cdot R\{f_2\} \cdot R\{f_3\} \cdot \ldots \cdot R\{f_{p+1}\}. \end{aligned} \tag{13}$$

That is the result must then also hold for p+1 polynomials.

Secondly, the result is proved for two polynomials. That is, we wish to show

$R\{f_1 \cdot f_2\} = R\{f_1\} \cdot R\{f_2\}$. Recall first from the definition, that for an n-degree polynomial $f(x)$, $R\{f(x)\} = x^n f(\frac{1}{x})$. Now suppose that $f_1(x)$ is of degree n and that $f_2(x)$ is of degree m. Then,

$$R\{f_1\} \cdot R\{f_2\} = x^{n+m} f_1(\tfrac{1}{x})\ f_2(\tfrac{1}{x}), \tag{14}$$

by definition of reverse. Now, let

$$f_1(x) \cdot f_2(x) \equiv \phi(x) \tag{15}$$

Thus

$$\begin{aligned} R\{f_1 \cdot f_2\} &= R\{\phi\} \\ &= x^{n+m}\, \phi\, (\tfrac{1}{x}), \end{aligned}$$

by definition of reverse; that is

$$R\{f_1 \cdot f_2\} = x^{n+m} f_1(\tfrac{1}{x}) \cdot f_2(\tfrac{1}{x}), \tag{16}$$

by (15). Hence, by (14)

$$R\{f_1 \cdot f_2\} = R\{f_1\} \cdot R\{f_2\} \qquad \text{QED.}$$

LEMMA 5: *An even-degree polynomial which is its own reverse is a symmetrical polynomial. That is, $f(z) \equiv R\{f(z)\} \Rightarrow f(z)$ is symmetrical.*

Proof: The proof follows easily from the definition of a symmetrical polynomial, and is therefore omitted.

THEOREM 2: *The determinant of $F(z)$, where $F(z)$ is the polynomial matrix as defined in equation (9), is a symmetrical polynomial of degree 2dD.*

Proof: $F(z)$ is a polynomial matrix whose ij-th element is a polynomial of (even) degree 2d given by equation (9), elaborated here for convenience.

$$f_{ij}(z) = \sum_{k=0}^{d-1} (K'_k)_{ij}\, z^k + \sum_{k=0}^{d} (K_{d-k})_{ij} z^{d+k}, \qquad i,j = 1, \ldots, D \tag{17}$$

where $(.)_{ij}$ denotes the ij-th element of the matrix in brackets.

Making use of the fact that $K_d = K'_d$, and that $(K'_k)_{ji} = (K_k)_{ij}$, $k = 0,1,2, \ldots, d-1$, by the definition of transposition, we have,

$$f_{ji}(z) = \sum_{k=0}^{d} (K_k)_{ij} z^k + \sum_{k=0}^{d-1} (K'{}_{d-k})_{ij} z^{d+k}, \qquad i,j = 1, \ldots, D \tag{18}$$

that is we have shown that $f_{ji}(z)$ is the reverse of $f_{ij}(z)$; that is $f_{ji}(z) = R\{f_{ij}(z)\}$, $i,j, \ldots, D$.

Now by definition of a determinant /6, pp. 225, 226/

$$|F(z)| = \sum_{k=1}^{D!} (\text{sgn})_k \prod_{i=1}^{D} f_{\sigma_k(i),i}(z), \tag{19}$$

where $\sigma_k(1), \ldots, \sigma_k(D)$ is the k-th permutation of the integers $1, \ldots, D$ where the sum is taken over all such permutations. The symbol $(\text{sgn})_k$ denotes +1 if the integers $\sigma_k(1), \ldots, \sigma_k(D)$ can be converted into $1, \ldots, D$ by an *even* number of interchanges; otherwise, $(\text{sgn})_k = -1$.

Note that every term in the summation consists of a product of D polynomials each of degree 2d. Consequently $|F(z)|$ is a polynomial of (even) degree 2dD, provided of course that K_0 is non-null, which we may assume without loss of generality.

Furthermore, every term in the summation is either symmetrical, or has its reverse in the summation. To prove this assertion, consider the k-th term in the summation:

$$(\text{sgn})_k \prod_{i=1}^{D} f_{\sigma_k(i),i}(z). \tag{20}$$

Since $f_{ji}(z) = R\{f_{ij}(z)\}$, $i,j = 1, \ldots, D$ and in view of Lemma 4, the reverse of the k-th term can be obtained quite simply by reversing the subscripts on the polynomials in the product and keeping the sign, to get:-

$$(\text{sgn})_k \prod_{i=1}^{D} f_{i,\sigma_k(i)}(z) \tag{21}$$

Now let $\sigma_{k'}(1), \sigma_{k'}(2), \ldots, \sigma_{k'}(D)$ be the permutation of $1,2, \ldots, D$ defined by the first subscripts on rearranging (21) so that the second subscripts are in the order $1,2, \ldots, D$. But $\sigma_{k'}(1), \sigma_{k'}(2), \ldots, \sigma_{k'}(D)$ is clearly another permutation of $1,2,\ldots,D$, with $(\text{sgn})_k = (\text{sgn})_{k'}$, and so,

$$(\text{sgn})_{k'} \prod_{i=1}^{D} f_{\sigma_{k'}(i),i}(z)$$

must be a term in the summation. If it is identical to the k-th term, then by Lemma 5 we have that the k-th term is symmetrical. If it is not identical to the k-th term, it is by construction, the reverse of the k-th term. Hence we have the result that every term in the summation in (20) is either symmetrical or has its reverse in the summation.

Therefore, by Lemmas 2 and 3, we have that the summation of 2dD-degree polynomials defining $|F(z)|$ is symmetrical, and of degree 2dD. QED.

THEOREM 3: *$|F(z)|$ possesses exactly dD roots, z_i, $i = 1, \ldots, dD$, some possibly repeated, satisfying $|z_i| < 1$, $i = 1, \ldots, dD$; provided only that there exists no root z such that $|z| = 1$.*[4]

Proof: By Theorem 2, we have that $|F(z)|$ is a symmetrical polynomial of degree 2dD, which is even.

By Theorem 1, therefore, $|F(z)|$ must possess exactly dD roots, z_i, $i = 1, \ldots, dD$, some possibly repeated, satisfying $|z_i|<1$; provided only that there exists no root z such that $|z| = 1$.
QED.

FINDING THE ROOTS

The computational efficiency of the Z-transform method of solution depends critically upon the method used to find the roots of $|F(z)|$. Hay and Holt /4, p. 242/ report the use of gradient methods. The efficiency of this approach is not discussed in that paper, but the authors seem to suggest /op.cit., p. 257/ that matrix iteration or matrix inversion are "... likely to be superior approaches ...". The method that is suggested in this paper[5] seems able to make the Z-transform method computationally superior to either matrix iteration or matrix inversion, and is developed in this section.

The method is as follows: To find the roots of $|F(z)|$, first form the 2dD×2dD matrix,

$$\kappa = \begin{bmatrix} \kappa_1 & \kappa_2 & \cdots & & \kappa_{2d} \\ -I & & & & \\ & -I & & & \\ & & \ddots & & \\ & & & -I & 0 \end{bmatrix}$$

$$\text{where } \kappa_i = \begin{cases} K_0^{-1} K_i, & i = 1, \ldots, d \\ K_0^{-1} K'_{2d-i}, & i = d+1, \ldots, 2d \end{cases}$$

and compute its 2dD *eigenvalues* using one of the several available computer programmes /9/ to do this. The eigenvalues of this matrix are the roots of $|F(z)|$. Note that this procedure requires that K_0^{-1} exists. We therefore wish to prove the following theorem.

THEOREM 4: *The roots of $|F(z)|$ are identical to the eigenvalues of κ, provided that K_0^{-1} exists.*

Proof: Substituting (7) into (6) and letting $T \to \infty$ as before, we have from the infinite-band portion:-

$$\sum_{i=0}^{d-1} K'_i \underline{y}^*_{t+i} + \sum_{i=0}^{d} K_{d-i} \underline{y}^*_{t+d+i} = \underline{k}_{t+d}, \quad t = 0,1,2, \ldots \tag{23}$$

where the time-partition $k' = [\underline{k}'_0, \underline{k}'_1, \underline{k}'_2, \ldots]'$ has been used. Note that (23) defines a set of *difference equations* of order 2d. F(z) is the *characteristic polynomial matrix* obtained from taking the Z-transform of this set of difference equations. To prove the theorem, (23) will be written as an equivalent *first*-order difference equation, whose characteristic polynomial is $(zI + \kappa)$. The roots of the *characteristic equation* obtained in either case must be identical, and in the latter case the roots of the characteristic equation are the eigenvalues of the matrix κ, by definition.[6]

Now, premultiplying (23) by K_0^{-1}, we get:-

$$\sum_{i=0}^{2d-1} \kappa_{2d-i} \underline{y}^*_{t+i} + \underline{y}^*_{t+2d} = K_0^{-1} \underline{k}_{t+d}, \quad t = 0,1,2 \ldots \tag{24}$$

Now, let

$$\underline{Y}_t^{(i)} = \underline{y}^*_{t+2d-i}, \quad i = 1,2, \ldots, 2d; \quad t = 0,1,2, \ldots \tag{25}$$

and define

$$Y_t = \begin{bmatrix} \underline{Y}_t^{(1)} \\ \underline{Y}_t^{(2)} \\ \cdot \\ \cdot \\ \underline{Y}_t^{(2d)} \end{bmatrix}, \quad t = 0,1,2, \ldots \tag{26}$$

The 2d-order difference equation in (24) can therefore be equivalently written as the following first-order difference equation:-

$$\kappa Y_t + Y_{t+1} = \Phi_t, \quad t = 0,1,2, \ldots \tag{27}$$

where we have let

$$\Phi_t = \underset{2dD\times 1}{\begin{bmatrix} K_0^{-1} \underline{k}_{t+d} \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix}}, \quad t = 0,1,2, \ldots \tag{28}$$

The characteristic equation of this system is

$$|zI + \kappa| = 0 \tag{29}$$

the solution to which is given by the eigenvalues of κ, by definition. But, from (23) the characteristic equation of the same system is given by

$$|F(z)| = 0$$

Hence, by a fundamental property of linear dynamic systems, we have that the eigenvalues of κ must be identical to the roots of $|F(z)|$. QED.

COMPARISON OF APPROACHES

In the Z-transform method suggested, the number of arithmetic operations required is dominated by two steps, viz., locating the eigenvalues of a 2dD×2dD matrix, and inverting a 2dD×2dD matrix. Each of these steps require in the order of $(2dD)^3$ arithmetic operations to be carried out, say $k(2dD)^3$ altogether, where k is an appropriate factor.[7]

Matrix iteration methods (e.g. Chow /2/) assume a finite horizon T. We can write T = nd, where n is a number chosen so that T approximates infinity, insofar as the convergence of the first-period solution is affected. (In practice this would seem to require n‡4, if the results in Theil /7, p. 167/ are any indication). The matrix iteration method requires a number of arithmetic operations dominated at each iteration by the multiplication of two $(d+1)D \times (d+1)D$ matrices. (This assumes that in the case of lagged variables, the state vector is augmented by these lagged variables so as to retain the first-order system dynamics assumed by Chow /2/, and that the state vector is of order not less than the decision vector). This multiplication requires in the order of $2(d+1)^3D^3$ arithmetic operations, and needs to be carried out nd times altogether. Matrix iteration therefore requires in the order of $2nd(d+1)^3D^3$ operations. This number of operations would therefore exceed that for the method suggested roughly whenever $2nd(d+1)^3 > 8kd^3$. Assuming n = 4, this is always true if $k < 6$, which will almost certainly be the case.

Matrix inversion (e.g. Theil /7/) would proceed by attempting to invert the large matrix in (7), for T large enough to approximate infinity and ensure convergence of the first-period solution. The number of arithmetic operations required to perform this inversion is in the order of $(DT)^3$. Writing T = nd, as before, this is in the order of $(ndD)^3$ operations.

Assuming n = 4, this number would exceed $k(2dD)^3$, the number required for the method suggested, whenever $k < 8$, which again will almost certainly be the case.

These are very rough calculations, but they indicate that the Z-transform method is likely to be computationally superior to either matrix iteration /1, p. 78 et.seq., 2/ or matrix inversion /7/. This is contrary to what Hay and Holt /4/ suggest, but is based on a different method of calculating the characteristic roots, than the gradient methods to which they allude /op.cit., p. 242/.

CONCLUSION

This paper has sketched an algorithm based on Z-transform methods, for the numerical solution of the optimization problem in which the objective function is quadratic, and the constraints are linear difference equations. It is shown that the method suggested is likely to be more efficient than either matrix inversion or matrix iteration.

Also included in this paper is a proof of a theorem guaranteeing the existence of the requisite number of roots of a characteristic polynomial satisfying conditions for the successful application of the Z-transform method to the problem. A recent proof by Hay and Holt /4/ uses a different approach and is perhaps not as rigorous.

The not too restrictive assumptions on which these developments have been based are that the span of cross-temporal effects be finite, and the system parameters be time-invariant. The other assumption that all parameters be real needs to be made in any case. A number of ancillary issues and extensions discussed in Hay and Holt /4, p. 249/ are not discussed here, specifically the treatment of complex roots, zero roots, time-varying parameters, and time-discounting of the objective function.

REFERENCES

/1/ BELLMAN, R.E. and KALABA, R.: *Dynamic Programming and Modern Control Theory*, New York: Academic Press, 1965.

/2/ CHOW, C.C.: "Optimal Control of Linear Econometric Systems", *International Economic Review*, 13 (1972), 16-25.

/3/ FRAZER, R.A., W.J. DUNCAN, A.R. COLLAR: *Elementary Matrices*, Cambridge University Press, 1938.

/4/ HAY, G.A. and HOLT, C.C.: "A General Solution for Linear Decision Rules: An Optimal Dynamic Strategy Applicable Under Uncertainty", *Econometrica*, 43 (1975), 231-259.

/5/ HOLT, C.C., F. MODIGLIANI, J.F. MUTH, and H.A. SIMON: *Planning Production, Inventories and Work Force*, Englewood Cliffs, N.J.: Prentice-Hall, 1960.

/6/ NOBLE, B.: *Applied Linear Algebra*, Englewood Cliffs, N.J.: Prentice-Hall, 1969.

/7/ THEIL, H.: *Optimal Decision Rules for Government and Industry*, Amsterdam, North Holland Publishing Co., North Holland Publishing Co., 1964.

/8/ THOMAS, S.F.: "A Quadratic Optimization Approach to Educational Enrolment Planning", (Unpublished) M.A. Sc. Thesis, Department of Industrial Engineering, Univ. of Toronto, 1974.

/9/ WILKINSON, J.H., C. REINSCH: "Linear Algebra", *Handbook for Automatic Computation, Vol. II* (Edited by F.L. BAUER et al), New York Heidelberg Berlin: Springer-Verlag, 1971.

ACKNOWLEDGEMENTS

I am deeply indebted to I.B. Turksen of the Department of Industrial Engineering, University of Toronto, for his helpful comments and criticisms during the preparation of the thesis on which this paper is based. That thesis was partially supported by a Norman Stuart Robertson Fellowship. I am also indebted to P.A. Morris and D. Beckles of the University of the West Indies, Mathematics Department, to C.C. Holt of Yale University, and to R.L. Graves of the European Institute for Advanced Studies in Management, all of whom made comments and criticisms of an earlier draft.

FOOTNOTES

[1] For a scalar sequence $(a_t | t = 0,1,2, \dots)$ the Z-transform of this sequence is defined as $Z(a_t) = \sum_{t=0}^{\infty} z^t a_t$ where z is a complex variable. The Z-transform of a vector sequence is defined as the vector whose elements are the Z-transforms of the individual elements of the original vector. See Frazer et al /3/ for a discussion of the algebra of Z-transforms.

[2] More precisely, $Z(\underline{k}_t) = \sum_{t=0}^{\infty} z^t \underline{k}_t$ is finite if the term $\underline{k}_t$ grows at a slower rate than the term z^t declines.

[3] In /8/ on which this section is based, the word "symmetric" was used. The term "symmetrical" is used here in a narrowly defined sense, without regard to any other meaning which might be attached to the word in the mathematics literature.

[4] It should be pointed out that the existence of a root z such that $|z| = 1$ is a very unlikely hair-line case, almost impossible to achieve in numerical computations.

[5] In the case where (23) were a *scalar* difference equation, the matrix κ would be known as the *adjacency matrix*, and Theorem 4 would be a known result. I am indebted to D. Beckles for pointing this out. The extension to vector difference equations implicit in Theorem 4, though somewhat straightforward, I haven't been able to find in the literature.

[6] First developed in /8/ (unpublished).

[7] It is well known that matrix inversion requires in the order of n^3 operations for a matrix of order n. See /6. p. 69/. The assertion that in the order of n^3 operations are required to locate the eigenvalues of a general matrix of order n, is based on an analysis of algorithms found in /9, contributions II/12, II/14/. Also, see /6, Ch. 10/.

THE STRUCTURE AND SOLUTION TECHNIQUES OF THE PROJECT INDEPENDENCE EVALUATION SYSTEM

FREDERIC H. MURPHY
FEDERAL ENERGY ADMINISTRATION

INTRODUCTION

The Project Independence Evaluation System (PIES) forecasts the state of the energy economy in selected future years (1980, 1985 and 1990) and reflects the impacts of a range of potential Federal policies on the prices paid for energy commodities, on the level of demands for these commodities, and on the level of imports of oil. The methodology used assumes that the role of government is to establish policies allowing participants in the economy to act in their own self-interest within the constraints imposed by these policies. The approach taken is to construct models for the different components of the energy system and then integrate the submodels or the outputs of the submodels into a forecast. This modularization allows for the ongoing improvement of the various segments of PIES without having to alter the entire system.

OVERVIEW

There is a set of supply models for each of the major raw materials, coal, oil and gas. They are built to simulate the response of the industry producing the raw material to increases and decreases in prices and are used to construct supply curves. Next, miniature models of refineries and electric utilities transform raw materials into consumable forms of energy. Estimates of the production capabilities of emerging technologies are added as well. The products consumed within the system are six petroleum products, gasoline, distillate, residual, jet fuel, liquid petroleum gases and other products from crude oil (lubes, waxes, etc.), and four other products, natural gas, electricity, bituminous coal and metallurgical coal. A large data base and set of econometric models are used to construct a demand model which estimates how the demand for each final product varies as the price of that product and the prices of other products change. As an example of how the price of one product impacts the demand for another, natural gas can be replaced by distillate for many industrial uses on approximately a BTU for BTU basis and vice versa. Therefore, if the per BTU price of one fuel gets out of line with the other and both are available, the lower-cost fuel is substituted for the higher-cost fuel. Since natural gas and distillate are not perfect substitutes, more fuel switching becomes economic and occurs in the model as the prices of the two commodities continue to diverge. The demand function is a log-linear approximation to a set of sector-specific demand models, that is,

$$\ln Q_j^k = a_j^k + \sum_{i=1}^{10} b_{ij}^k \ln p_i^k \quad \text{for } j = 1,\ldots,10.$$

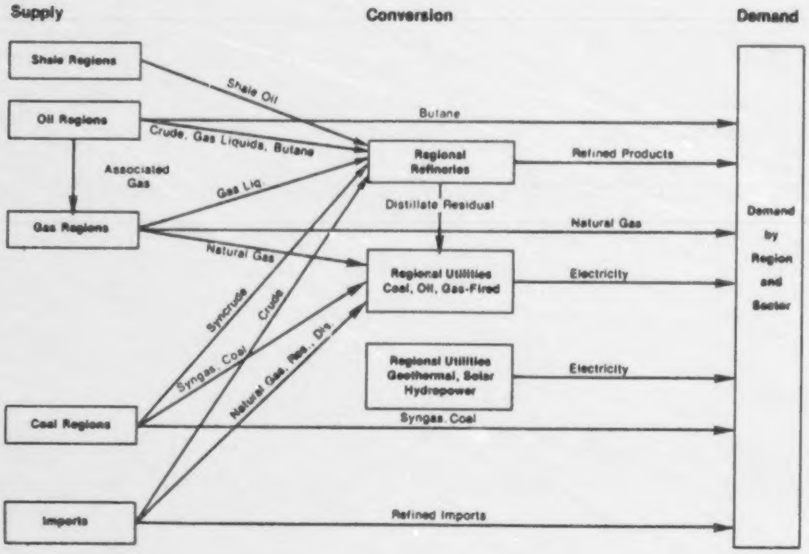
where Q_j^k is the quantity of product j demanded in sector k when fuels sell at the retail prices p_i^k. The sectors are household, commercial, raw material, industrial and transportation. When i=j, b_{ij}^k is referred to as the own elasticity for product i and is negative. If $i \neq j$, then b_{ij}^k is a cross elasticity and in our case is positively signed. With all other prices constant, an x percent change in p_i^k leads to an xb_{ij}^k percent change in the demand for product j. A full discussion is contained in (2), describing each segment of the demand model in detail.

Demands are forecasted for a larger slate of products than is available from the supply structure. Also, the supply prices are wholesale as opposed to retail prices. To evaluate the demands for the ten supplied products at the supply (wholesale) prices, the following steps are taken. First, each demand product is associated with one of the supply products, e.g., petroleum coke with other petroleum products. Markups appropriate for going from wholesale to retail prices in the given sector are then added to the wholesale prices and the demand equations for each fuel are evaluated. The resultant quantities are then aggregated across the appropriate demand products and across sectors to determine the demand for the supply product at the wholesale price.

The various components of the system are tied together by a transportation network that moves raw materials or products from where they are produced to where they are consumed or where they are used to produce other energy products. The flows within the system are shown in Figure 1.

FIGURE I

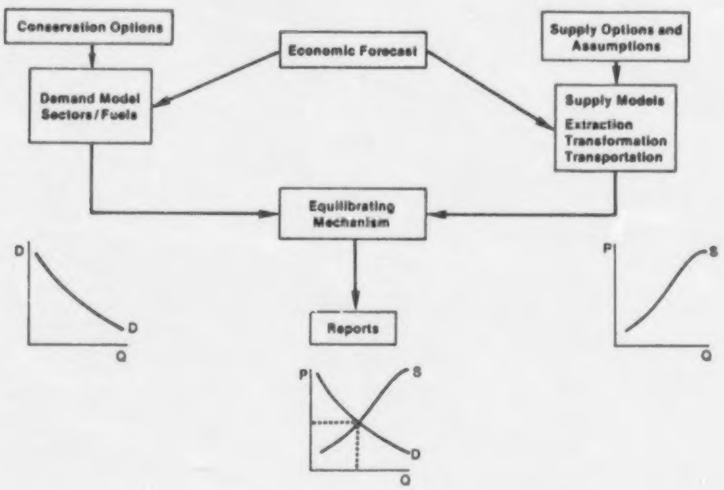
Flow of Materials


Supply
Conversion
Demand
Shale Regions
Shale Oil
Oil Regions
Butane
Crude, Gas Liquids, Butane
Associated Gas
Regional Refineries
Refined Products
Gas Liq.
Distillate Residual
Gas Regions
Natural Gas
Natural Gas
Regional Utilities Coal, Oil, Gas-Fired
Electricity
Syncrude
Crude
Syngas, Coal
Natural Gas, Res., Dis.
Regional Utilities Geothermal, Solar Hydropower
Electricity
Coal Regions
Syngas, Coal
Imports
Refined Imports
Demand by Region and Sector

The supply functions, the demand function, and the conversion activities are combined by the PIES integrating system. The output of the integrating system is a general equilibrium solution (Figure 2) of the mathematical representation of the energy economy, i.e., a set of balanced supplies and demands as well as market clearing prices for each fuel is provided. Formally stated, the problem is to find a vector of prices $\bar{p}$ such that the vector of demands $\bar{D}(\bar{p}) = \bar{S}(\bar{p})$ the vector of supplies at $\bar{p}$.

FIGURE 2

Integrating Model Framework


Conservation Options
Economic Forecast
Supply Options and Assumptions
Demand Model Sectors/Fuels
Supply Models Extraction Transformation Transportation
Equilibrating Mechanism
Reports
D
D
Q
P
S
Q
P
S
D
Q

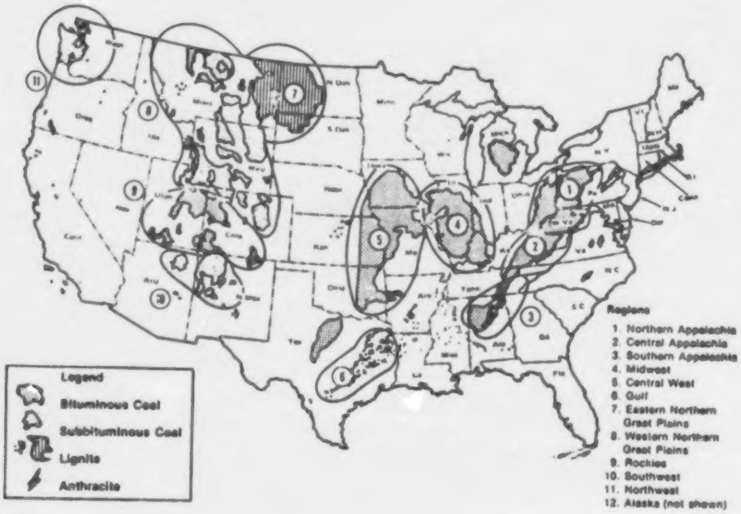
REGIONAL STRUCTURE

Each raw material has a regional structure in the model that represents the unique characteristics of its resource base. There are also specific regional definitions for conversion and demand activities. The purpose of all of the regional detail within the model is not to provide results for regional analyses but to provide better national figures. Western low sulfur coal would implicitly be used in New England, for example, if transportation cost differentials were ignored. Also, more nuclear plants would be built if a national average transportation cost were used for coal.

There are twelve coal regions which are chosen so that each is relatively compact and contains only a few coal categories of bituminous coal. Some regions also contain metallurgical coal. Coal is separated into five BTU categories, and each of the bituminous categories is divided into three sulfur types: high, medium and low sulfur. The coal regions are shown in Figure 3. Transportation is a substantial part of the costs in using coal. Within the model, the more compact the coal region, the better the estimate of transportation costs from the coal region to the utility or demand regions. As a consequence, even though some regions such as Central and Southern Appalachia contain the same categories of coal, they are modeled as two separate regions to have a better estimate of transportation costs. To further refine the costs of shipping, a transshipment network is used. Coal moves from the coal region to a collection of transshipment nodes: cities such as Cincinnati, New Orleans and Atlanta. Instead of shipping the coal to the demand regions from the transshipment points, it is shipped to a selected set of cities in the demand region with the demand in each city a fixed fraction of the demand region needs.

FIGURE 3

PIES Coal Supply Regions


Legend
Bituminous Coal
Subbituminous Coal
Lignite
Anthracite
Regions
1. Northern Appalachia
2. Central Appalachia
3. Southern Appalachia
4. Midwest
5. Central West
6. Gulf
7. Eastern Northern Great Plains
8. Western Northern Great Plains
9. Rockies
10. Southwest
11. Northwest
12. Alaska (not shown)

There are 12 oil and 13 gas regions based on National Petroleum Council (NPC) regions and special Alaskan Regions. The refinery regions are Petroleum Allocation for Defense Districts or PADD's. The crude oil (condensates, etc., from gas regions) are moved into the refinery regions from the oil and gas regions by pipeline or tanker. And the six product groups are moved from the refinery regions to either the utility or demand regions.

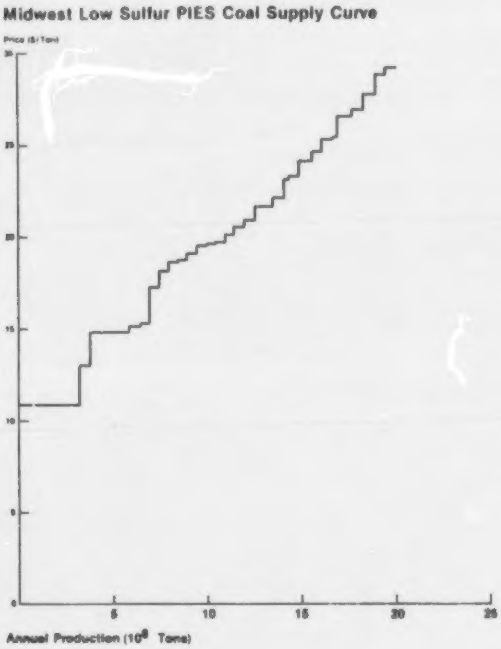
For ease in modeling, the utility and demand regions are the same. They are FEA regions. Unlike the regions for supplying other forms of energy, a utility region may serve only the corresponding demand region except for the shipment of hydroelectric power from the Northwest to California. This greatly simplifies the calculating of the average cost of electricity, that is, the price of electricity to consumers.

SUPPLY ACTIVITIES

The traditional approach used by economists is to estimate output as a function of capital and labor without serious regard to the resource base (see, for example [2]. This is inappropriate here because the most important factor affecting the supply of fuels is the character and extent of the resource base. Rather than using historical time series data and statistical techniques to directly predict future raw material and product availability, operations research-based process models are built to simulate the actual production capabilities given the resources of an energy sector.

The supply models are used to construct supply curves that are step-function approximations to continous functions. For example, each step of the coal supply curve for each region represents the annual rate of production from a specific mine type within two mine classes, surface and deep. In Figure 4 the lowest-cost steps on the coal supply curves are associated with existing mines or mines that are about to be opened. Here the capital is sunk or mostly sunk and the mines will be operated as long as the marginal revenue is at least equal to the operating cost. The higher-cost steps ensure the capital recovery necessary for opening a new mine. There are supply curves for each oil and gas region that distinguish primary, secondary and tertiary production. Different crude types such as West Coast heavy and Wyoming Mix are distinguished by region. Each crude type is produced in proportion to its historic share for the region.

FIGURE 4

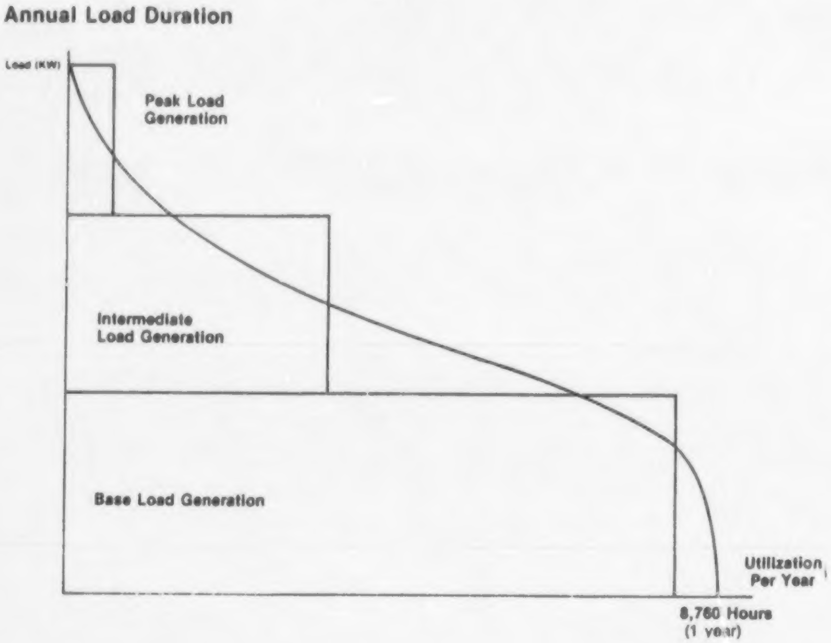


CONVERSION ACTIVITIES

The electric utility and refinery sectors of the PIES system are embedded directly into the general equilibrium model. In this respect, the conversion models are different from the supply models for coal, oil and gas where only the outputs (supply curves) are directly included in the general equilibrium model.

The key to modeling electric utilities is that they cannot inventory their product and must produce electricity on demand. This means that utilities must own some equipment that runs most of the time and some equipment that runs only during peak demand periods. The demand levels for electricity during a year are represented by the load duration curve. A point (x, y) on the load duration curve in Figure 5 shows that for x hours during the year at least y kilowatts were demanded. This curve, for modeling purposes, is divided into three pieces: base, intermediate, and peak. The kinds of generation equipment that can be used include nuclear, hydroelectric, coal-fired (with and without scrubbers), residual fired, natural gas fired, and simple-cycle and combined-cycle distillate plants as well as new technologies. Any of these types of equipment, other than nuclear, can operate in base, intermediate, or peak, whichever is most economic for the utility. Nuclear operates in base only.

FIGURE 5



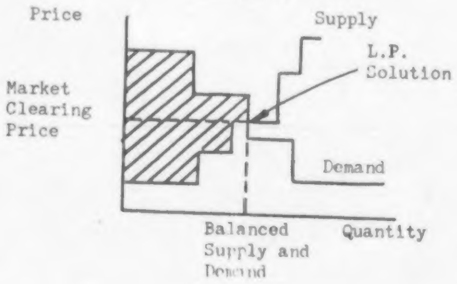
The major role of the refinery sector is to reflect the appropriate relative prices for different crude types based on crude oil attributes and the appropriate relative prices for the four product classes based on product characteristics. What should happen is that factors such as the quantities of the four products consumed and the quantities of the different types of crude oils should interact in the refinery sector based on relative demands and availabilities. Crude oils with greater yields of the more heavily demanded products should be more valuable and the products that dominate demand, such as gasoline, should be more expensive.

A primary product yield from each crude oil is estimated using historical data. Next, by parametrically varying the yields of a large-scale refinery linear program, the costs of shifting the product slate are determined. The result is a model that is an extreme point representation of a refinery.

THE INTEGRATING MECHANISM

The solution procedure involves inserting a step-function approximation to the demand function into a linear program containing the supply curves and the models of conversion activities. The approximation to the demand function ignores the effects of the price of one product on the demands for other products, e.g., only the natural gas price affects natural gas demand. The linear program is then solved with the objective of maximizing the area under the difference between the demand and supply curves. This is mathematically equivalent to finding where the supply and demand curves intersect (Figure 6). How close the prices and quantities are to being on the demand function containing the cross price effects is then measured. If the linear programming quantities are not within one percent of the demand function quantities evaluated at the prices taken from the linear program, the equilibration process continues with a new demand demand function approximation. The demand function containing the cross price effects is evaluated at an average between the prices from the linear program and the previous estimate of the equilibrium prices; and a new approximation to the demand curve is constructed around this point.

FIGURE 6



In the past, people have tried to find economic equilibria by just inserting a single demand point and successively replacing the demand quantities with new values from the demand model evaluated at the dual variables. The dual variables seemed to oscillate and not converge. By inserting a demand curve approximation, an equilibrium is achieved after 6 to 8 iterations involving the solution of linear programs. Currently, we vary all prices the same percentage simultaneously in calculating the step widths. If the convergence were slower, we would try varying all prices with percentages proportional to a trajectory of prices from successive iterations. Other procedures have been considered but not tested.

THE RELATION OF PIES TO GENERAL EQUILIBRIUM THEORY

The Neoclassical model of exchange may be described as follows. Each consuming unit i for i=1,..., m has an initial endowment of assets:

$$W^i = (W_1^i \ldots, W_n^i).$$

At a price vector $\pi = (\pi_1, \ldots, \pi_n)$, consuming unit i demands a vector of products $d_i(\pi)$ and has an income of $I^i = \sum_{j=1}^{n} \pi_j W_j^i$. For convenience, $\sum_{j=1}^{n} \pi_j = 1$ is usually added as a requirement, i.e., the analysis is in terms of constant dollars. Since in equilibrium an individual cannot spend more than the revenue from the sale of his assets,

$$\pi d_i(\pi) = \pi W^i.$$

This leads to Walras law:

$$\pi \sum_{i=1}^{m} d_i(\pi) = \pi \sum_{i=1}^{m} W^i,$$

That is, the monetary value of what is demanded equals that of what is supplied. Letting $g_j(\pi) = \sum_{i=1}^{m} (d_i^j(\pi) - W_j^i)$ be the excess demand function for asset j, for j=1,...,n an economy is in equilibrium when

$$g_j(\pi) \leq 0$$

$$\pi_j \, g_j(\pi) = 0.$$

That is, either supply equals demand or supply exceeds demand and the price of the asset is zero. When the pure exchange economy is generalized to include activities for the conversion of one asset into another using linear activities, we have the following definition of a competitive equilibrium:

Definition - A price vector π^* and a vector of activity levels y^* constitute a competitive equilibrium if:

a. Supply equals demand in all markets, or $d(\pi^*) = By^* + W$ where B is the matrix of possible activities; and

b. production is consistent with profit maximization in the sense that $\sum_i \pi_i^* b_{ij} \leq 0$ with equality if $y_j^* > 0$.

Part b is a requirement that excess profits are associated only with rents on scarce resources.

In PIES the assets are raw materials such as coal, oil and biomass as well as electrical generation equipment, refineries, pipelines, etc. Added to these is an aggregate asset which represents capital for new equipment and for developing new resources, labor and other nonenergy resources. The components of this aggregate asset are priced in terms of 1975 dollars, giving us the objective function cost coefficients in the linear programming subproblem. The number of assets in PIES is large because each step on each regional supply curve of a fuel is a different asset in the exchange economy with a different equilibrium price (dual variable on the bound row). Every bounded variable is an asset, as is every product in every demand region. In PIES an economic equilibrium is found where all assets are priced relative to the aggregate asset. The prices may then be normalized to achieve $\sum_{j=1}^{n} \pi_j = 1$. This leads to an alternative interpretation of the PIES mechanism for searching for an equilibrium. The goal at each iteration is to satisfy Walras law with the dual variables as prices and a successively improved approximation to the demand model.

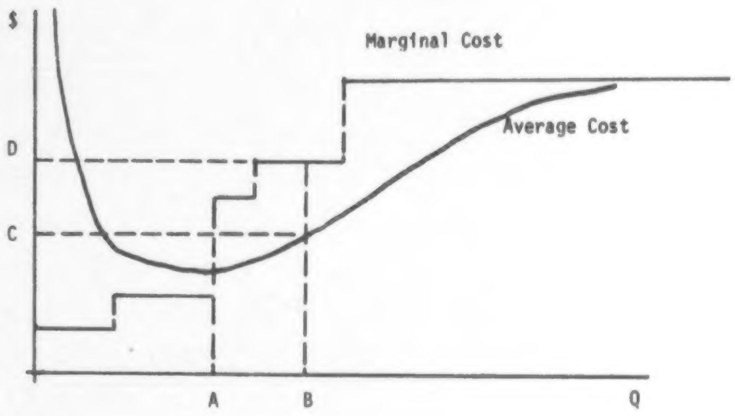
NONCOMPETITIVE PHENOMENA MODELED IN PIES

There are three areas where regulatory actions that alter the structure of the economy are modeled. These are the average cost pricing of electricity, current oil import entitlements and interstate regulation of natural gas.

UTILITY REGULATION

Since there are increasing returns to scale to power transmission and distribution, the electric utilities constitute a natural monopoly which must be regulated in some fashion. Currently, public utility commissions regulate utilities to provide them with a reasonable rate of return on their total investment. This means that customers are charged the average, not marginal, cost of delivering electricity. The cost curves for delivery look as follows:

FIGURE 7



Given a set of fuel prices, the marginal cost curve is a step function where each step represents bringing into use a different kind of generation equipment. The higher costs are due to using less efficient equipment or including the capital costs for acquiring new equipment. For low quantities of electricity the average cost curve is higher than the marginal cost curve. This is because the cost of the existing equipment is included in the average cost of electricity.

The prices from a linear program are marginal prices. Therefore, for the demand model to respond to average instead of marginal prices an adjustment must be made to the linear program. The approach taken is to approximate the average cost curve with the marginal cost curve. This is done in the following manner.

Say the amount of electricity demanded in a given region at the end of a linear programming step is B. The difference between the marginal and average costs of electricity is C-D. In revising the linear program for the next iteration the transmission cost is changed so that the marginal cost is approximately equal to the average cost at B. Letting α be the adjustments to the true transmission cost in the linear program from the previous iteration, the new adjustment is $(C-D + \alpha)/2$. The average is taken for smoothing purposes.

This average cost pricing mechanism can have a substantial effect on convergence. First, if the quantity of electricity is less than A, then a decreasing average cost curve is approximated by a nondecreasing function. The result is a perverse behavior where the amount of electricity generated decreases at each iteration while the cost increases. We have also experienced another form of non-convergent behavior because of our implementation. We have seen the quantity of electricity oscillate between two steps on the marginal cost curve, causing the correction between average and marginal costs to fluctuate and not stabilize. This means that the marginal to average cost adjustment is fluctuating, leading to an oscillation in prices and quantities. This occurred in the Northwest where there was a big jump in marginal cost in going from hydro power to fossil and/or nuclear power as the marginal source of electricity. We do not have a complete explanation of this phenomenon. Our best guess is that it has to do with the behavior near the minimum of the average cost curve of our implementation of this approach to average cost pricing.

OIL ENTITLEMENTS

The current regulations on oil production require that the average price of domestically produced oil be below $8.00. Since the marginal price of oil in the United States is the world price because of our import dependence, the marginal price of oil products would be refining costs plus crude oil costs at the world price if there were no other provisions in the law. The regulations specify that refiners who use domestic oil pay $X per barrel into a fund from which refiners who use imported oil receive $Y per barrel. This entitlement to the users of imported oil makes them indifferent between domestic and imported oils, that is,

$$P_D + X = P_I - Y$$

where P_D is the average domestic oil price and P_I is the import price. It is also important that the fund never run deficits or surpluses, that is,

$$XQ_D = YQ_I$$

where Q_D is the domestic production and Q_I is imports. There also are entitlements to different types of domestic oil. There is a legal definition of what is called "old" oil and "new" oil with "old" oil having a substantially different price from "new" oil.

Entitlements are modeled as follows. The total domestic oil production given the regulations is estimated. The supply curves for oil in the linear programming matrix are replaced by this supply point and the cost in the objective function is the average price for this oil. New activities are added to the matrix that essentially tax domestic oil an amount X and give a credit of Y to imported oil. The X and Y are recalculated between L.P. iterations using the above equations. In equilibrium we, therefore, have a scheme where these equations are satisfied and a forecast of the impact of this form or regulation.

NATURAL GAS REGULATION

Currently, there are two markets for natural gas: the unregulated intrastate markets and the Federally regulated sales of natural gas across state boundaries. By law the interstate gas price is based on historical costs; this has lead to a two tier market and has developed where intrastate gas sells at a higher price than interstate gas. Consequently, the only new sales of interstate gas are coming from onshore areas where fields are close to interstate pipelines and there are no intrastate pipelines nearby or from the outer-continental shelf which is under Federal jurisdiction. The supply of interstate natural gas in a given year can be estimated by taking the current rate of production and reducing it by the natural decline from existing wells in the onshore regions and adding to this the production from the outer-continental shelf. Since the customer must be charged the average cost of interstate gas (there are three price levels for various vintages of domestically produced interstate gas plus the costs of liquefied natural gas (LNG) and imports from Canada), there is a shortfall in supply in regions where there is little or no intrastate gas available.

To deal with the shortfall a priority scheme has been developed by the FPC to allocate gas to states by classes for a given pipeline. Each state then allocates the gas available from the pipeline. Each state then allocates the gas available from the pipeline to customers within the state based on its own priority structure.

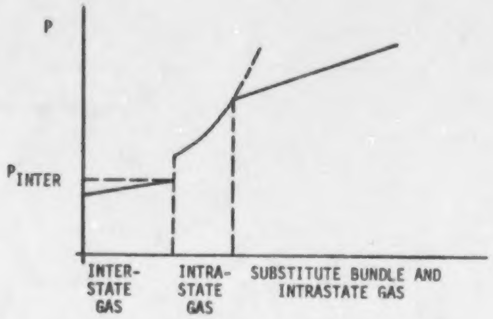
The modeling approach taken is to assume that gas customers fall into two distinct groups: those with interstate gas and those that must use intrastate gas or a bundle of other fuels with the bundle containing such fuels as electricity and distillate. To determine the customers who receive interstate gas, the available domestically produced gas is allocated in an approximation to the FPC priority scheme across the nation rather than by individual pipeline. The priorities, in order, are residential, commercial, raw material and industrial. After a region receives its share of domestically produced gas, imported gas available to the region is then rolled in until either all demand is satisfied or there is no more gas available. Excess demand in curtailed sectors is satisfied first by intrastate supply to the extent it is available at creditable prices and then by a displacement or switching of this demand to other fuels in a way that is sensitive to other fuel prices, sector-specific end use efficiencies and historical shares.

Say for illustration residential demand is satisfied but only α of commercial demand is satisfied and there is no raw material or industrial demand to be satisfied.

95

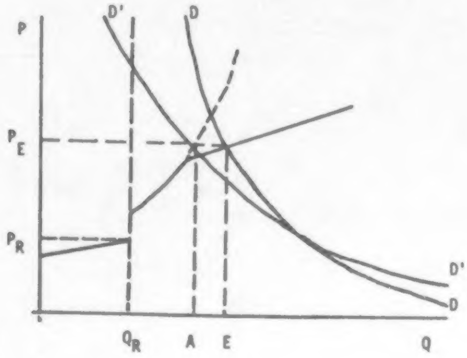
The total supply to all commercial customers is

FIGURE 8



The demand for natural gas is the sum of the demand satisfied by interstate gas at the interstate gas price plus the demand for intrastate gas and the substitute fuel bundle. Ignoring cross elasticities the demand for gas is $Q = a\,P^{\varepsilon}$ where P is the natural gas price, ε is the elasticity of natural gas demand and and a is a constant. At the interstate price P_R, the quantity of gas demand met, Q, is $\alpha a P_R^{\varepsilon}$. This means that the demand curve for intrastate gas and the substitute bundle is $(1-\alpha)aP_R^{\varepsilon}$. In Figure 9 the demand curve for gas is D'. Because the inter- and intrastate markets are separate, the total demand for natural gas is represented by the demand curve D where $Q = Q_R + (1-\alpha)aP^{\varepsilon}$. That is, the quantity of intrastate gas and the substitute fuel bundle demanded at P_E is $E - Q_R$. The quantity of intrastate gas consumed is $A - Q_R$ and the quantity of demand met by the fuel bundle is E-A. The price of the gas displacement bundle is a sector-specific market price constructed from its fuel components. As the diagram shows, this activity is assumed to dominate the further movement up the interstate supply curve. The assumption which justifies this is that the process model of fuel substitution on the demand side is more reliable than the econometric model at very high prices.

FIGURE 9



Note that the total quantity of gas demanded with this demand curve is greater than the quantity demanded with the original demand curve D'. This is because consumers of interstate gas see the low interstate gas price and not the higher intrastate price.

COMPARISONS OF PIES TO OTHER PROCEDURES FOR FINDING A COMPETITIVE EQUILIBRIUM

The only other technique for finding a competitive equilibrium with computational results involves discrete approximations of the simplex of prices and a search for a fixed point. The fixed point approach of Scarf [3] with enhancements by Hanson [3] has results that cannot be seriously compared with PIES because the problems solved are on a much smaller scale. Appendix 2 in [3] gives computational experience for Scarf's algorithm. He estimates that computation time varies as m^4. On an IBM 360-50 he estimates the time to solve a 15 asset problem to be about 2 minutes. Extrapolating, a 100 asset problem would then take more than 2,000 minutes. The solution step in PIES with 100 products consumed, not counting intermediate products or raw materials which are in the thousands, takes from 20 to 30 minutes on a 370-168 under MVS with the number of linear programs to be solved ranging from 6 to 8. When natural gas regulation is in the model the number of iterations increases to 15-20. The reason for the increase in iterations is that supply of intrastate natural gas is inelastic in many regions leading to large price changes for little quantity change.

BIBLIOGRAPHY

1. Federal Energy Administration, 1976 National Energy Outlook, FEA-N75/713.

2. MacAvoy, P. W. and R. S. Pindyck, "Alternative Regulatory Policies for Dealing with the Natural Gas Shortage," The Bell Journal of Economics and Management Science, Autumn, 1973.

3. Scarf, H., Computation of Economic Equilibria, Yale University Press, New Haven, Connecticut, 1973.

97

NATIONAL AND INTERREGIONAL PROGRAMMING MODELS OF LAND AND WATER USE AND THE ENVIRONMENT

by

Earl O. Heady, Kenneth J. Nicol, and Dan Dvoskin
The Center for Agricultural and Rural Development
Iowa State University, Ames, Iowa

Abstract

A set of large-scale programming models has been developed and quantitatively applied to analyze land use, water use, environmental restraints and impacts, agricultural and food policies, export problems, income redistribution patterns and other facets of the nation's agricultural sector. Perhaps these are the largest operationally useful mathematical programming models in existence. The models incorporate nine land class restraints in each of 223 producing regions. They include 51 water resource regions with corresponding restraints and 35 market regions with demand relations for all relevant endogenous commodities. The models generate results at national, state, regional or local levels. Their results have been used by all major national commissions on food policy, water allocation, and land use.

Many applications of mathematical programming models have now been made. We have completed and underway a set which is rather unique in its size, scope and national policy uses. While we employ both linear and quadratic specification, main reliance is placed on linear models. These national and interregional models applied to agriculture and natural resources have been under evolution for a dozen years. This time period has allowed us to build up a vast data bank and add many dimensions to models. At various stages in their development, the models have been the main quantitative base for various commissions dealing with national policy. They have served this role for the President's Food and Fiber Commission [3], the National Water Commission [4], the Water Resources Council's National Water Assessment [7], and the Midwest Governors Conference on Land Use [5] and for numerous applied studies for the Environmental Protection Agency [6], the National Water Quality Commission [9], and the Soil Conservation Service [10].

In this paper, we report on one set of models capable of evaluating water and land use and their impacts on the environment at both national and regional levels [8]. While we apply one such model to evaluate potentials in environmental improvement through controlling soil loss or sedimentation from farm land, the general model set is capable of analyzing many other facets of resource use and environmental impacts as these relate to trade, agricultural or other policies. We also have completed models which incorporate legislative or price restraints to attain energy conservation [1], to promote environmental quality through reduced use of nitrogen fertilizer and pesticides [2], which enhance the environment through stream flow regulations to protect wildlife habitats and others.

The models have the capacity to evaluate simultaneously variables and outcomes in (a) national markets, prices, incomes and employment and (b) production patterns, resource use and economic structure of rather small resource regions, under the posed imposition of alternative policies or futures. We believe that models with these characteristics and capabilities are extremely important for the future as national, state and local entities evaluate and consider implementation of environmental, land use, water and other resource and technological restraints related to problems emerging under the nation's advanced state of economic development. Otherwise, the programs and policies imposed by states, municipalities and regional planning bodies will encounter unexpected economic effects, causing them to be nullified because they give inequitable distributions of the costs and benefits of the goals attained. For example, initial solutions of our models pose the certainty that individual states which impose restrained patterns of land use, water runoff, sedimentation and technologies will find that, through market impacts, producers and resource owners of other states and location will realize economic gains while those of the imposing state will bear the costs in lower incomes and reduced resource prices. Even for certain quality controls imposed at the national level, relative returns can be positive in some regions and negative in other regions.

Models Reviewed

The models reviewed in this paper encompass the whole of U.S. agriculture and the land and water use relating thereto. These demand-allocation models incorporate all major agricultural quality interaction reflecting restraints in resources for 223 agricultural producing regions (Figure 1), soil characteristics in 1,891 land resource groups, water resources for 51 water supply

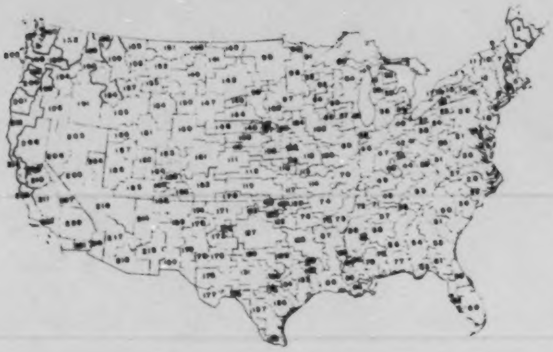


Figure 1. The 223 producing areas

regions (Figure 2), in the 17 Western States, and demand or commodity balances in 30 consumer market regions (Figure 3). The models, which incorporate a transportation submodel for commodities and water and product transfer activities, allow selection of optimal resource use patterns and environmental quality impacts for the nation in future time periods. They also reflect comparative advantage in the allocation of land and water to competing alternatives as represented in relative yields, general technologies, environmentally restrained technologies, production costs, transport costs and imposed environmental restraints. They allow substitution of land at one location for water at another location a thousand miles away (or vice versa). Similarly, they allow and analyze these substitutions when environmental restraints are applied to restrain the technologies used in any one resource region. Finally, they allow evaluation of various policy alternatives in use of land and water resources, and environmental quality controls in interaction with commercial agricultural policies, export goals and domestic demands in both regional and national markets.

Objectives

Our overall objectives in building these models are to determine (a) whether the nation has enough land and water resources to meet domestic and export food needs when various environmental quality restraints are imposed, (b) the optimum spatial allocation, for the nation and internally for each individual producing, land and water region, of these resources accordingly, (c) the extent to which sacrifices must be made in environmental quality goals as other goals (food prices, exports, treasury costs, farm income, energy use, resource values, income distribution, etc.) are attained--or vice versa, (d) the cost to regions and the nation as various land use patterns and environmental quality goals are attained, (e) the optimal selection among alternative producing technologies and land use patterns, for each region and for the nation, as various environmental quality restraints are imposed, and (f) miscellaneous impacts including those relating to farm size and income, the

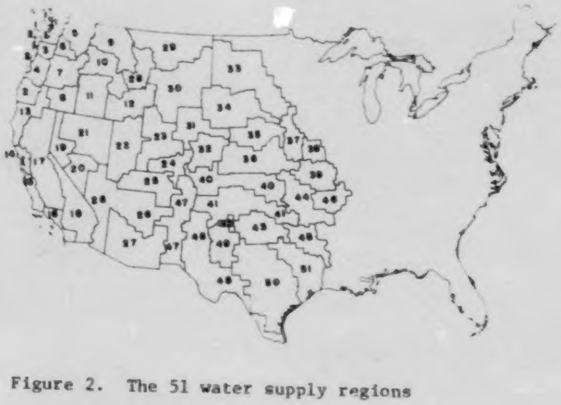


Figure 2. The 51 water supply regions

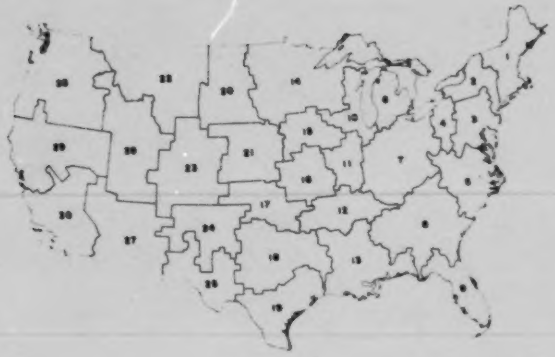


Figure 3. The 30 consumer market regions

distribution of the costs and benefits of these patterns or allocations, employment and income generation in rural areas.

Environmental quality controls based on runoff and sediment transport, for example, will have direct impact on crops produced and technology used on erodible lands (e.g., slope of land and amount and intensity of rainfall). However, these environmental restraints also will have "chain reactions" in optimal land use in regions 100 or 1,000 miles away which are not subject to runoff and sediment loss since, nationally, a new configuration of comparative advantages will be created in relation to both the environmental restraints and commodity and resource markets Land with characteristics giving rise to runoff and sediment loss may be required to shift to more forages, livestock production or forest products--although the outcome also will depend on row crop yields, costs and returns under mechanical erosion control practices. In the "chain reaction" or regional interdependence relationships, land at one distant location not subject to erosion which once produced cotton may now optimally be allocated to soybeans to meet national demands while at a different location, nonerodible land once allocated to wheat now may be specified for feed grains as the national livestock ration and export demands are met. Restraints on chemical fertilizers and pesticides have similar remote and complex interactions in resource use among the many different land regions of the nation. Generally, those regions of ample rainfall and irrigation water will be shifted towards a less intensive use of land while adapted regions with less runoff and relatively less dependence on imported technological inputs will tend towards more intensive land use (e.g., more grain and less forage and livestock production).

Because of regional interdependencies, it is impossible to plan nationally efficient uses of land and environmental quality controls on a region-by-region or regionally independent basis. The models used incorporate interdependencies among the hundreds of land resource regions and allow for both direct and indirect impacts among regions whether they are contiguous or distant from each other. Not only do they need to incorporate these interdependencies among land resource regions, water regions and market regions, but also they need to allow them among resources and commodities. They need to allow substitution of land at one location for water at a different and distant location, since a policy restraint which limits the use of capital technology on land at one location can be offset by a reallocation or extended use of water at another location--or vice versa. Great flexibility prevails in the national livestock ration, the major demand determinant in national land use, and shifts in or restraints on land use can allow or cause limits on grain production in one location to be replaced by soybean, forage or a substitute feed grain in another location. An efficient land use-environmental quality model needs to allow all of these interdependencies over the nation with reflection back to optimal land employment for each land resource region.

Nature of Models

To illustrate the general nature of the models involved, under the restraint of presentation space and time, we use a model projecting to the year 2000 for a population of 284 million, free market conditions with trend levels of agricultural technology in each of 223 agricultural producing regions. This model, only one in a series we are building, emphasizes optimal land use patterns, agricultural water allocation, agricultural technology and soil conservation methods under environmentally restrained soil loss. (Subsequent formulations of the model set includes environmental limits on chemical fertilizers, pesticides and livestock wastes.) The objective function in equation (1) minimizes the cost of producing and transporting the various crop and livestock commodities among producing and land resource regions of origin, regions of processing, and regions of consumption. The costs allow the system to consider different technologies (cropping or land use systems and mechanical practices) in restraining soil loss to alternative environmental quality levels. The costs of water consumption and transfer also are included in equation (1). The programming prices and costs cover all factor costs (except land rents which are reflected in shadow prices) and thus allow simulation of a long-run market equilibrium for each commodity with a national allocation reflecting the comparative advantage of each of the 223 producing regions, subject to environmental restraints and the level and location of consumer demands. The objective function is minimize OF where:

$$\begin{aligned} OF = & \sum_i (\sum_k (\sum_m X_{ikm}\ UC_{ikm} + \sum_n Y_{ikn}\ UC_{ikm} + \sum_m Z_{ikm}\ UC_{ikm}) \\ & + \sum_p L_{ip}\ UC_{ip} + DPP_i\ UC_i + IPP_i\ UC_i + DWH_i\ UC_i \\ & + IWH_i\ UC_i + FLG_i\ UC_i + FP_i\ UC_i) + \sum_w (WB_w\ UC_w \\ & + WD_w\ UC_w + WT_w\ UC_w) + \sum_t \sum_c T_{tc}\ UC_{tc} \end{aligned} \qquad (1)$$

where the variables, parameters, and other terms are defined in a following section.

Restraints and variables

Each land group has alternative crop management systems producing commodities with associated yields and soil loss subject to the soil types, average weather prevailing and the conservation tillage practices utilized. Data were developed in conjunction with the Soil Conservation Service, U.S.D.A., to represent soil loss per acre under various mechanical practices and rotations or land use systems. The soil loss alternatives are evaluated through the universal soil loss equation (2). The equation used for each crop management system is of the form

$$SL = K \cdot L \cdot S \cdot R \cdot C \cdot P \qquad (2)$$

where

SL is the per acre gross soil loss;
K is the erodibility factor associated with the soil type;
L is the computed value relating slope length to soil loss control;
S is the derived from a nonlinear function relating slope gradient to level of soil loss;
R is an index of erodibility for the rainfall of the area accounting for varying levels of intensity, duration and measured rainfall;
C is an adjustment factor giving an index of the relative ability of alternative cropping patterns to reduce soil loss;
P is an adjustment factor to account for the potential soil loss reduction from adopting conservation practices.

Each activity in the model represents an alternative crop management system which incorporates a given rotation, crop tillage method and a conservation practice for an individual land resource group. The rotation and tillage method combine to give unique C value and the conservation practice determines the P factor. The K, L, and S factors are dependent on the soil characteristics and the regional rainfall patterns determine the R factor.

Associated with the alternative crop management systems are specific per acre crop costs and crop yields for each region. The cost data reflect expenditures on machinery, labor, pesticides, nonnitrogen fertilizers (nitrogen is balanced endogenous to the model), and miscellaneous production items. The component costs reflect different efficiencies of farming resulting from working land in straight rows, contours, strip cropping, with terraces, under minimum tillage or under conventional tillage. The alternative costs also reflect the higher pesticide requirements and lower machinery and labor requirements for crops under a reduced tillage cultivation pattern. The costs sum to an aggregate which depends on the particular cropping management system and when combined with the outputs from the system, reflect the comparative advantage of each system on each land class in each region.

The outputs from the system reflect yields of each crop and the associated quantity of soil loss per acre per year. The interaction within the system also is reflected in a nitrogen balance subsector where the nitrogen flows in the model are examined. The entire cost and yield section of the model is interlocked with alternative technologies, levels of resource input and alternative input uses to meet domestic and export demands. The nitrogen-fertilizer-crop yield section is an example of other interrelationships in the model. Nitrogen available in each region is an independent variable in the crop yield equation but the source of the nitrogen may vary. It can be supplied from chemical fertilizers, livestock wastes and from nitrogen fixation by legumes. The livestock wastes available are dependent on the type and quantity of feed available for livestock and the concentration of the animals in the region. Also affecting the yields of the crop is the land class on which it is grown and the conservation and tillage practice associated with the cropping management system.

Both dryland and irrigated crop variables are included for producing regions in the 17 Western States which grow irrigated crops and the model allows selection among dryland or irrigated farming for each region. A range of livestock rations (variables) is allowed in all 223 producing regions since the least-cost feed mix can be drawn from various grain, forage and pasture crops grown in the region or imported (where allowed) from others. The model includes variables representing various cropping systems and technologies affecting soil loss, livestock production, commodity transportation, water transfers, consumer demand fulfillment and alternative export levels.

Each of the 223 producing regions has land restraints of the nature indicated in equations 3-9. Each region has a soil loss restraint as in equation (10), a nitrogen balance equation as in (11) and a pasture restraint as in (23). Each of the 51 water supply regions has a water restraint as in equation (13), where variables and parameters are defined subsequently.

Each of the 30 consuming regions has net demand equations, for all of the relevant crop and livestock activities as illustrated by equation (14). Regional consumer demand quantities were determined exogenously from geographic and national projections of population, economic activity, per capita incomes and international exports through the region for 2000. National demands were defined for cotton and sugar beets as indicated in equation (15). Poultry products, sheep, and other livestock were regulated at the consuming region level. International trade was regulated at the regional levels as indicated in equations (16) and (17).

Commodities included in the endogenous analysis are soil loss, nitrogen, water, corn, sorghum, wheat, barley, oats, soybeans, cotton, sugar beets, tame hay, wild hay, improved pasture, unimproved and woodland pasture, cropland pasture, public grazing lands, forest lands grazed, all dairy products, pork, and beef. Also accounted for prior to solution of the model are other crops including fruits, nuts, vegetables, rice, flax and broilers, turkeys, egg production, sheep and other livestock.

Dryland cropland restraint, each region by land class:

$$\sum_{m} X_{ikm}\, a_{ikm} \leq LD_{ik} \qquad (3)$$

Irrigated cropland restraint, each region by land class:

$$\sum_{n} Y_{ikn}\, a_{ikn} + \sum_{m} Z_{ikm}\, a_{ikm} \leq LR_{ik} \qquad (4)$$

Dryland wild hay restraint, each region:

$$DWH_{i}\, a_{i} \leq ADWH_{i} \qquad (5)$$

Irrigated wild hay restraint, each region:

$$IWH_{i}\, a_{i} \leq AIWH_{i} \qquad (6)$$

Dryland permanent pasture restraint, each region:

$$DPP_{i}\, a_{i} \leq ADPP_{i} \qquad (7)$$

Irrigated permanent pasture restraint, each region:

$$IPP_{i}\, a_{i} \leq AIPP_{i} \qquad (8)$$

Forest land grazed restraint, each region:

$$FLG_{i}\, a_{i} \leq AFLG_{i} \qquad (9)$$

Soil loss restraint, each region, each land class, each activity:

$$SL_{ikm+n} \leq ASL_{ikm+n} \qquad (10)$$

Nitrogen balance restraint, by region:

$$FP_{i} + \sum_{p} b_{ip}\, L_{ip} + EL_{ibi} - EC_{ifi} - \sum_{k}(\sum_{m} X_{ikm}$$

$$f_{ikm} + \sum_{n} Y_{ikn}\, F_{ibn} + \sum_{m} Z_{ikm}\, f_{ikm}) - DPP_{i}\, f_{i}$$

$$- IPP_{i}\, f_{i} - DWH_{i}\, f_{i} - IWH_{i}\, f_{i} - FLG_{i}\, f_{i} = 0 \qquad (11)$$

Pasture use restraint, each region:

$$\sum_{k}(\sum_{m} X_{ikm}\, r_{ikm} + \sum_{n} Y_{ikn}\, r_{ikn} + \sum_{m} Z_{ikm}\, r_{ikm}) + DPP_{i}$$

$$r_{i} + IPP_{i}\, r_{i} + FLG_{i}\, r_{i} - \sum_{p} L_{ip}\, q_{ip}$$

$$- EL_{i}\, q_{i} \geq 0 \qquad (12)$$

Water use restraint, by water region:

$$WB_{w} \pm WT_{w} \pm WI_{w} - WO_{w} - WX_{w} - WE_{w} + WD_{w} - \sum_{i\epsilon w}$$

$$IWH_{i}\, d_{i} - \sum_{i\epsilon w} IPP_{i}\, d_{i} - \sum_{k} \sum_{i\epsilon w} (\sum_{m} X_{ikm}\, d_{ikm} +$$

$$\sum_{n} Y_{ikn}\, d_{ikn} + \sum_{m} Z\, F_{ikm}\, d_{ikm}) - \sum_{i\epsilon w} \sum_{p} L_{ip}\, d_{ip}$$

$$- \sum_{i\epsilon w} PN_{i}\, d_{i} \geq 0 \qquad (13)$$

Commodity balance restraint, each consuming region:

$$\sum_{k} \sum_{i\epsilon j} (\sum_{m} X_{ikm}\, cy_{ikmc} + \sum_{n} Y_{ikn}\, cy_{iknc} + \sum_{m} Z_{ikm}$$

$$cy_{ikmc}) \pm \sum_{i\epsilon j} \sum_{p} L_{ip} cy_{ipc} \pm \sum_{t\epsilon j} T_{tc} \pm E_{jc} - \sum_{i\epsilon j}$$

$$PN_{i}\, cy_{ic} - EL_{j}\, cy_{jc} \geq 0 \qquad (14)$$

National commodity balance restraints, for cotton, sugar beets and spring wheat:

$$\sum_{i} \sum_{k}(\sum_{m} X_{ikm}\, cy_{ikmg} + \sum_{n} Y_{ikn}\, cy_{ikng} + \sum_{m} Z_{ikm}\, cy_{ikmg})$$

$$- \sum_{i} PN_{i}\, cy_{ig} - EX_{c} \geq 0 \qquad (15)$$

National export restraints:

$$\sum_{j} E_{jc} \geq EX_{c} \qquad (16)$$

National import restraints:

$$\sum_{i} E_{ic+e} \leq IM_{c+e} \qquad (17)$$

Non-negativity restraints:

$$X_{ikm},\ Y_{ikn},\ Z_{ikm},\ L_{ip},\ DWH_{i},\ IWH_{i},\ DPP_{i},\ IPP_{i},$$

$$FLG_{i},\ FP_{i},\ EL_{i},\ WB_{w},\ WT_{w},\ WI_{w},\ WD_{w},\ WX_{w},\ WE_{w},\ PN_{i},$$

$$T_{tc},\ E_{jc},\ E_{icre} \geq 0 \qquad (18)$$

The subcripts and variables for the above equations are defined in the section below.

Subscripts and variables of the model

The subscripts and variables relating to the equations in the text are those which follow:

subscripts

$c = 1,2,\ldots,15$ for the endogenous commodities in the model,
$e = 1,2,\ldots,5$ for the exogenous livestock alternatives considered,
$g = 1,2,3$ for the commodities balanced at the national level,
$i = 1,2,\ldots,223$ for the producing areas of the model,
$j = 1,2,\ldots,30$ for the consuming regions of the model,
$k = 1,2,\ldots,9$ for the land classes in each producing area,
$m = 1,2,\ldots,$ for the dryland crop management systems on a land class in a producing area,
$n = 1,2,\ldots,$ for the irrigated crop management systems on a land class in a producing area,
$p = 1,2,\ldots,$ for the livestock activities defined in the producing area,
$t = 1,2,\ldots,458$ for the transportation routes in the model,
$w = 1,2,\ldots,51$ for the water supply regions in the model.

variables

a, the amount of land used by the associated activity from the land base as indicated by the subscripts;
AIPP, the number of acres of irrigated permanent pasture available in the subscripted producing area;
ADWH, the number of acres of dryland wild (noncropland) hay available in the subscripted producing area;
AFLG, the number of acres of forest land available for grazing in the subscripted producing area;
ADPP, the number of acres of dryland permanent pasture available for use in the subscripted producing area;
AIWH, the number of acres of irrigated wild (noncropland) hay available in the subscripted producing area;
ASL, the per acre allowable soil loss subscripted for land class, producing area and activity;
b, the units of nitrogen equivalent fertilizer produced by livestock, subscripted for producing area and activity;
cy, interaction coefficients (yield or use) of the relevant commodity as regulated by the associated activity and specified by the subscripts;
d, the per unit of activity water use coefficient as regulated by the associated activity and specified by the subscripts;
DPP, level of use of dryland permanent pasture in the subscripted producing area;
DWH, level of use of dryland wild (non-cropland) hay in the subscripted producing area;
E, level of net export for the associated commodity in the associated region as specified by the subscripts;
EC, level of exogenous crop production by subscripted region;
EL, level of exogenous livestock production consistent with the subscripted region;
EX, the level of national net export for the subscripted commodity as determined exogenous to the model;
f, the units of nitrogen equivalent fertilizer required by the associated activity and specified by the subscripts;
FLG, level of forest land grazed in the subscripted producing area;
FP, number of pounds of nitrogen equivalent fertilizer purchased in the subscripted producing area;
IM, level of national net imports for the subscripted commodities as determined exogenous to the model;
IPP, level of use of the irrigated permanent pasture in the subscripted region;
IWH, level of use of the irrigated wild (non-cropland) hay in the subscripted region;
L, level of the livestock activity with the type and region dependent on the subscripts;
LD, number of acres of dryland cropland available for use as specified by the region and land class subscripts;
LR, number of acres of irrigated cropland available for use as specified by the region and land class subscripts;
PN, level of population projected to be in the subscripted region;
q, units of pasture, in hay equivalents, consumed by the associated livestock activity and specified by the subscripts;
r, units of aftermath or regular pasture, in hay equivalents, produced by the associated cropping or pasture activity and identified by the subscripts;
SL, level of soil loss associated with any activity over the range $m+n$ in the region and land class designated by the subscripts;
T, level of transportation of a unit of the commodity either into or out of the consuming region designated by the subscripts;
WB, level of water purchase for use in the water balance of the water supply region designated by the subscript;
WD, level of desalting of ocean water in the water supply region designated by the subscript;
WE, level of water to be exported from the water supply region subscripted;
WI, level of movement of water in or out of the water supply region through the interbasin transfer network;
WO, level of water requirement for onsite uses such as mining, navigation and estuary maintenance in the water supply region subscripted;
WX, level of water use for the exogenous agricultural crops and livestock in the water supply region subscripted;
X, level of employment of the dryland crop management system, rotation, in the region and on the land class as designated by the subscripts;
Y, level of employment of the irrigated crop management system, rotation, in the region and on the land class as designated by the subscripts;
Z, level of employment of the dryland crop management system, rotation, on the land class

in the region as designated by the subscripts when the land has been designated as available for irrigated cropping patterns.

Illustration of Results

Solution of the model provides indication of optimal land use in each producing region or each of the 1,891 land resource groups at prescribed levels of environmental quality restraints, consumer demand and distribution, export levels and other policy, or market and technology parameters. Our illustration is in the case where the only environmental restraint is soil loss. It also designates the level of production in each region and the optimal flows of commodities to consuming regions and export markets. For purposes of illustration, we refer to solutions where (a) soil loss is not restricted and (b) soil loss is restricted to 5 tons per acre per year for each of the 1,891 soil resource groups and exports are at a modest level. While land use could be mapped or indicated by each of the 1,891 land resource groups, we illustrate on the basis of the 223 producing regions only. The model indicates not only land devoted to each crop use in each region and group, but also can indicate technologies for each such as dryland or irrigated, alternative rotations, conventional or reduced tillage methods and others which affect land and water use and sedimentation.

Figure 4 indicates an optimal distribution of row crop acreage among the 223 producing regions, Figure 5 indicates an optimal distribution of the close grown crops and Figure 6 gives the hayland distribution when no restraints are placed on soil loss or chemical nitrogen use.

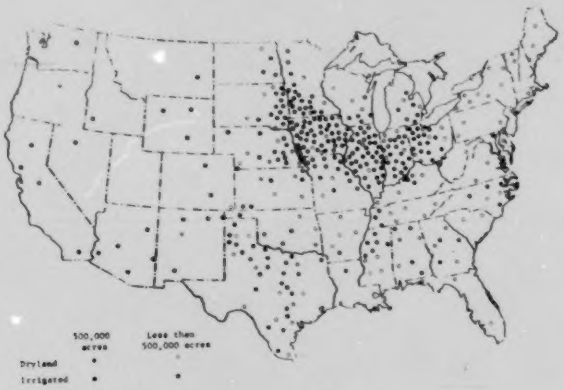


Figure 4. Location of dryland and irrigated row crops with no soil loss restriction in 2000.

Soil loss

While land use, tillage methods and soil loss are generated in the models by producing regions and land resource groups, we summarize results only for seven major geographic regions of the U.S. because of time and space restraints. While solutions of the model were made for several soil loss, export and nitrogen restriction levels, we similarly summarize solutions only for two soil loss levels, one export level and unrestrained nitrogen use (except for nitrogen balance within a producing region).

Restricting soil loss per acre to five tons would distribute land use and technologies interregionally to reduce national soil loss to 727 million tons. Without the restriction, interregional land use allocations and technologies to meet export demands would generate a national soil loss 3.5 times greater, or 2,677 million tons. As Table 1 indicates the reduction in average per acre soil loss, as a source of sedimentation, would be extremely large on land classes V-VIII which are most erosive. While we do not do so here, our models allow indication of soil loss changes by each individual region.

Regional variation in reduced soil loss per acre is great. Largest reductions take place in the South Atlantic (18.2 tons per acre) and South Central (11.5 tons per acre) regions where land and current land use methods give rise to serious erosion (Table 2). The reduction in soil loss when a 5 ton per acre limit is imposed is attained especially by a switch from conventional tillage-straight row farming to contour, strip-cropping and terraces (Table 3). There also is a significant shift to reduced tillage farming practices to attain the environmentally attained soil loss of five tons per acre. Acres receiving reduced tillage practices increase from around 21 million in the unrestricted solution to near 58 million acres under the five-ton solution. Conventional tillage practices decline from 248 million acres under the unrestricted soil loss to 201 million acres under the solution for a five-ton soil loss. Within the conventional tillage group,

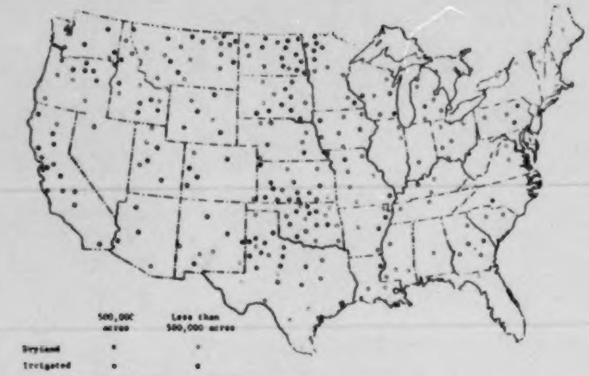

500,000 acres
Less than 500,000 acres
Dryland
Irrigated

Figure 5. Location of dryland and irrigated close grown crops with no soil loss restriction in 2000.

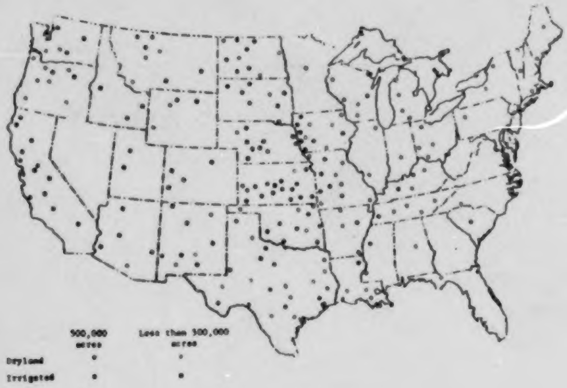

500,000 acres
Less than 500,000 acres
Dryland
Irrigated

Figure 6. Location of dryland and irrigated hay with no soil loss restriction in 2000.

Table 1. National soil loss total and average per acre by land resource groups for two levels of soil loss restriction, 2000.

Item	Unrestricted soil loss	5 ton soil loss
Total tons (million ton)	2677	727
Average tons per acre		
class I & II land	6.2	2.7
IIIE & IVE land	17.8	3.1
other III & IV land	15.6	2.8
V - VIII land	28.5	1.5
national average	9.9	2.8

Table 2. Average per acre soil loss by major region for two levels of soil loss restriction models, 2000.

Region	Unrestricted soil loss	5 ton soil loss
National	9.9	2.8
North Atlantic	9.0	3.5
South Atlantic	21.5	3.3
North Central	9.2	2.8
South Central	15.1	3.6
Great Plains	3.2	1.5
North West	2.3	1.7
South West	3.3	2.6

Table 3. Thousand acres of cultivated land by conservation - tillage practices for two levels of soil loss restriction, 2000.

Conservation tillage	Unrestricted soil loss	5 ton soil loss
Conventional tillage	247,894	201,238
straight row	233,475	129,120
contoured	11,254	37,116
strip cropped & terraced	3,165	35,002
Reduced tillage	21,219	57,644
straight row	21,219	24,822
contoured	0	18,902
strip cropped & terraced	0	13,920

straight-row farming is nearly halved. Contouring is tripled and strip cropping-terracing practices are increased 1,000 percent to meet soil loss restrictions (Table 3). While reduced tillage nearly triples and very large increases occur in contouring, terracing and strip cropping, straight-row methods of reduced tillage do not increase importantly.

The shift in acreages (Table 4) is partly hidden in the reduction of 16.5 million acres used for grain crops and a corresponding increase of only 5.5 million acres in hay on cultivated lands (Table 4). Part of the production required to meet national demands comes from an increase in noncropland roughage production (permanent hay and pasture). More of the reduced acreage required to meet the demand for agricultural products, results because of the shift in production to the higher cost and higher yielding erosion control practices. Also a shift in acreage between land classes puts the grain crops on the higher yielding and less erosive lands.

Costs of production, in conjunction with the transportation network and the soil loss restrictions imposed, determine the national equilibrium prices for the commodities. Table 5 indicates the relative equilibrium prices of the commodities generated by the model under the two levels of soil loss and a single export level. Soil loss restrictions have the largest effect on prices for commodities which concentrate on land with high soil loss potential. Compared to absence of soil loss restrictions, cotton and soybean prices increase over 20 percent while wheat and hay crops increase by less than 10 percent. The increase in grain prices result in corresponding increases in cattle prices. In evaluating the effect of any environmental policy alternative, the effect on the desired parameter and the change in farm price of agricultural products are two easily observed changes in our models.

Changes summarized at the national level do not, of course, reflect the effects in particular regions and on individual enterprises. These, however, are all available from our models. The shift in production from one region to another results in income repercussions on the rural community affected. The effect of such a shift depends on the degree of multiple level resource use.

The data in Tables 1-5 indicate that American

REFERENCES

1. Dvoskin, D. and E.O. Heady. U.S. Agricultural Production Under Limited Energy Supplies, High Energy Prices, and Expanding Exports. CARD Report 69. Center for Agricultural and Rural Development, Iowa State University, Ames, Iowa, Nov. 1976.

2. Dvoskin, D. and E.O. Heady. U.S. Agricultural Export Capabilities Under Various Price Alternatives, Regional Production Variations, and Fertilizer-Use Restriction. CARD Report 63. Center for Agricultural and Rural Development, Iowa State University, Ames, Iowa, Dec. 1975.

3. Heady, E.O. and L.V. Mayer. Food Needs and U.S. Agriculture in 1980. National Advisory Commission on Food and Fiber, Washington, D.C. Aug. 1967.

4. Heady, E.O., H.C. Madsen, K.J. Nicol, and S.H. Hargrove. Agricultural Water Policies and the Environment. CARD Report 40T. Center for Agricultural and Rural Development, Iowa State University, Ames, Iowa, June 1972.

5. Huemoeller, W.A., K.J. Nicol, E.O. Heady, and B. Spaulding. *Land Use: Ongoing Developments in the North Central Region.* Center for Agricultural and Rural Development, Ames, Iowa, Nov. 1976.

6. Madsen, H.C., K.J. Nicol, and E.O. Heady. Environmental Impacts and Costs in Agriculture in Relation to Soil Loss Restrictions and Nitrogen Fertilizer Limitations. Report to the U.S. Environmental Protection Agency. Center for Agricultural and Rural Development, Iowa State University, Ames, Iowa, Nov. 1973.

7. Meister, A.D., E.O. Heady, K.J. Nicol, and R.W. Strohbehn. U.S. Agricultural Production in Relation to Alternative Water, Environmental, and Export Policies. CARD Report 65. Center for Agricultural and Rural Development, Ames, Iowa, June 1976.

8. Nicol, K.J., and E.O. Heady. *A Model for Regional Agricultural Analysis of Land and Water Use, Agricultural Structure, and the Environment: A Documentation.* Center for Agricultural and Rural Development, Ames, Iowa, July 1974.

9. Nicol, K.J. and E.O. Heady. The Economic Impacts of Abatement Control of Irrigation Return Flows and Pollutant Run-off from Nonirrigated Agriculture. PB 248 807. National Technical Information Service. Springfield, Virginia, July 1975.

10. Nicol, K.J., E.O. Heady, and H.C. Madsen. Models of Soil Loss, Land and Water Use Spatial Agricultural Structure, and the Environment. CARD Report 49T. Center for Agricultural and Rural Development, Ames, Iowa, July 1974.

11. Wade, J.C. and E.O. Heady. A National Model of Sediment and Water Quality: Various Impacts on American Agriculture. CARD Report 67. Center for Agricultural and Rural Development, Iowa State University, Ames, Iowa, July 1976.

Table 4. National production of row crops, close grown crops and rotation roughage crops for two levels of soil loss restriction, 2000.[1]

Land use	Unrestricted soil loss	5 ton soil loss
Acres cultivated (000)	269,113	258,882
Row crops (000)	148,226	136,035
Close grown crops (000)	75,535	73,478
Rotation roughage crops (000)	45,352	49,369
Non-rotation roughage crops (000)	303,060	310,697

[1] Demand levels are based on projected per capita food consumption levels, 284 million people in 2000 and international trade of grains equal to the 1969-1971 annual averages.

Table 5. Relative farm level prices for some agricultural commodities with two levels of soil loss restriction, 2000.

Commodity	Unrestricted soil loss	5 ton soil loss
Corn	100	107
Wheat	100	103
Soybeans	100	115
Cotton	100	112
Hay	100	101
Cattle	100	104
Hogs	100	105
Milk	100	100

agriculture has great capacity and flexibility in adapting to certain environmental quality goals. By shifting land use among the many producing regions and land resource groups in terms of their comparative advantage in yields, commodity costs, location, and transportation, national and regional demands can be met without large increases in food prices and costs for consumers at the export level examined. The level of exports per se may have greater impact on consumer food costs than does a relatively wide adaptation of agriculture and land use to environmental quality goals. We will, however, provide quantitative analysis of these possibilities, along with other environmental quality practices, in upcoming presentations.

A NONLINEAR PROGRAMMING APPROACH
TO PREFERENCE MAXIMIZED MENU PLANS

Joseph L. Balintfy
and
Prabhakant Sinha
University of Massachusetts
Amherst

I INTRODUCTION

Human diet problems fall into two major categories known as food planning and meal planning problems. Both problem areas have been shown to be amenable to mathematical definition, formulation and solution.

Food planning is concerned with decisions as to which food entity, and how much, to purchase subject to given budgetary, nutritional, and acceptability requirements. The first statement of this problem in the context of the cost of subsistence is due to Stigler [17]. This was later reformulated by Dantzig [10] as the classical example of a linear programming model, and was refined later with respect to consumer acceptance by Smith [16]. The concept is still in use in terms of food groups in connection with USDA family food plans [15].

Meal planning is a decision problem of finding an optimum sequence of meals consisting of combinations of prepared foods, called menu items, to be eaten by a person or a population, such that the required structure of the meals and given budgetary, nutritional and food production specifications are met. The entities of meal planning are menu items of known portion size, with the food ingredients per portion defined by the respective recipes. This problem was first identified and solved as a mathematical programming problem by Balintfy [2]. The first approach and some of the later refinements [3] were considering one meal (or day) at a time, using what is now called a multistage menu scheduling algorithm. Each meal was a least-cost (best buy) combination of items selected on the basis of avoiding incompatibility of items between meals and within meals. The former rule was effected by requiring a minimum separation of meals between consecutive appearance of the same item or the same kind of item on the schedule. The latter rule was observed by restricting items from the same attribute class to occur in the solutions.

The concept of minimum separation of items was not only useful in menu scheduling as a safeguard for variety, i.e., acceptability, but it could also be used to establish upper bounds on the frequency of items in a given time period. This realization led to the development of a bounded linear programming model to meal planning [5] which defined in a single stage the optimum (least cost) frequencies of menu items for a period, called a menu plan, which later could be scheduled into a sequence of meals.

Initially, both versions of these modeling approaches to meal planning had cost-minimizing objective functions and assured acceptability operationally only by variety and entry restrictions. The cost minimization objective was selected primarily to show the economic impact of mathematical optimization as opposed to conventional methods. Another rationale for cost minimizing was the paucity of data and methodology to represent food preferences quantitatively as meal planning objectives. Indeed, cost savings from 10-30 percent of food cost have been achieved in a variety of applications [9,11].

These applications were initiated in hospitals, although a variety of other institutional feeding programs, such as school lunch service, college food service, as well as nursing home, detention home, and military food service operations also could utilize a better approach than the prevailing conventional method of menu planning. Such methods cannot take into consideration explicitly and quantitatively the population preferences, the nutrient composition and the cost of menu items, and hence the resulting conventional menus are ipso facto suboptimal and often infeasible relative to the stated objectives and constraints [3].

The role of food preferences has been long recognized, and food preference and preferred serving-frequency data have been routinely collected in the food service industry. Such information was, however, utilized only subjectively in menu planning, mostly because the functional relation between the preference for an item and its serving frequency was not recognized. Benson [8] was the first who represented this relation by fitting data to a polynomial function, but no attempt was made to use his results either in menu planning or in mathematical programming formulations.

Major developments in the last years have taken place in the mathematical modeling of food preferences by the discovery of the existence of time-related preference functions. The impact of this new development on the modeling of meal planning decisions as mathematical optimization problems is investigated in the sections that follow.

II MATHEMATICAL FORMULATION OF FOOD PREFERENCE FUNCTIONS

It is assumed - as is observable in reality - that most food is consumed in discrete portions at discrete points in time. Consequently, one can relate a measure of satisfaction with the event that a fixed quantity of food is consumed at a given time. One can also assume that for foods that are familiar and are more or less routinely consumed by an individual, satisfaction with foods is not only experienced but also anticipated to such a degree that a measure of utility can be elicited by collecting preference ratings for a set of foods on some centered scale. In this investigation, therefore, it will be assumed that preference ratings are estimates of the measure of satisfaction of an individual with a food. The word food is used here as a collective term applicable to the special cases of both food groups and menu items as the case may be.

Let $h(t)$ be the preference rating of an individual for a given food item at time t where t is measured from the last time when the item was consumed. Clearly $h(t)$ is a function for which the following properties can be postulated:

(a) $h(0) = -\infty$; since zero time interval between eating a fixed portion of a food is impossible as well as intolerable.

(b) $h(\infty) = a$; where "a" is a positive constant expressing the preference for a known food item when it has not been available (hence not consumed) for a long time. It is possible that "a" is itself some function of absolute time reflecting shifts of taste or seasonality, but these effects are not considered here.

(c) $h[\alpha t_1 + (1-\alpha)t_2] \geq \alpha h(t_1) + (1-\alpha)h(t_2)$ for $0 \leq \alpha \leq 1$ and $0 < t_1 < t_2 < \infty$; which means that the preference-time function is assumed to be concave. The evidence for this assumption is indirect, but convincing. The concavity of $h(t)$ is consistent with the observation that most people tend to separate their preferred food items on the time scale with fairly equal time intervals as opposed to clustering them on a succession of meals.

(d) $\frac{d}{dt}[h(t)/t] = 0$ at some unique value of $t = T_o$, $(0 < T_o < \infty)$ where $h(t)/t = g(t)$ is the preference function averaged over time, and it is postulated that this time-average has a unique maximum at time T_o. This property of $g(t)$ is supported by the evidence that people can estimate values of T_o by the ability of responding to questions such as "how frequently do you like to eat this given item?"

Empirical verification of the above assumptions is available in a report: Modeling Food Preferences Over Time [6]. It was found that preference for a particular menu item can be best described by the recursive formula

$$(1)\quad h(t_n) = f(t_n - t_{n-1}) - e^{-r(t_n - t_{n-1})}[f(\infty) - h(t_{n-1})]$$

where $r > 0$ and t_n indicates the absolute time scale when the item was consumed n times before. Figure 1 shows the analytical form of this recursive relation in terms of parameters estimated from observed data.

Here

$$(2)\quad f(t_n - t_{n-1}) = f(t') = a - be^{-ct'}$$

where $a > 0$, $b > 0$, and $c > 0$ are parameters of a first order differential equation which postulates that the rate of increase of preference in time is proportional to the effect of "monotony" expressed by the $[a - f(t')]$ difference.

Substituting (2) into (1) and letting

$$\lim_{nt_n \to \infty} h(t_n) = h(t) \quad ,$$

(1) reduces to

$$(3)\quad h(t) = a - \frac{be^{-ct}}{1 - e^{-rt}}$$

This is the expression of preference-time relations with the assumption that the item is repeatedly consumed at identical t time intervals. From (3) the time averaged function of preference is obtained as

$$(4)\quad g(t) = \frac{a}{t} - \frac{be^{-ct}}{t(1 - e^{-rt})}$$

Figure 2 shows the shapes of the $f(t)$, $h(t)$ and $g(t)$ functions for a particular item as rated by one subject. It is seen that $f(t) > h(t)$ for all values of t and $g(t)$ has a unique maximum at $T_o = 5$ days.

The authors' earlier report [6] has shown the methods and the results of estimating the parameters of the $h(t)$ function from questionnaires. It is noted, however, that one of the four parameters, r, is needed basically only for the computation of the recursive relations in (1). For considerations where the time intervals can be regarded as equidistant, a direct analytical expression for $h(t)$ can be attempted with more economy in parameters. For this reason an approximation of $h(t)$ by $H(t)$ is introduced in the form

$$(5)\quad H(t) = a - \frac{1}{ut^v} \quad .$$

It is obvious that $H(t)$ satisfies conditions (a) through (d) stipulated for preference-time functions. Moreover, it is possible to estimate the parameter of $H(t)$ from that of the $h(t)$ function. Conditions (a) and (b) are clearly satisfied at the same value of "a" for both functions. By imposing two additional conditions, the two other parameters of the $H(t)$ function can be uniquely defined. These two conditions are as follows:

1. Requiring that both functions have identical zero crossing, i.e.

$h(t) = H(t) = 0$ at $t = t_o$

This is satisfied if

$$(6)\quad a = \frac{be^{-ct_o}}{1-e^{-rt_o}} = \frac{1}{ut_o^v}$$

The value of t_o can be determined by solving the implicit nonlinear equation

$$\phi(t_o) = a(1-e^{-rt_o}) - be^{-ct_o} = 0$$

by some numerical analytical method.

2. Requiring that the time averaged preferences have maximum at the same $t = T_o$ time for both functions. As in expression (4), $G(t)$ is defined as

$$(7)\quad G(t) = \frac{a}{t} - \frac{1}{ut^{v+1}} \quad .$$

Then the condition that

$$\frac{d}{dt}G(t) = \frac{d}{dt}g(t) = 0 \quad \text{at} \quad t = T_o$$

implies that

$$\frac{-a}{T_o^2} + \frac{v+1}{uT_o^{v+2}} = 0$$

i.e.

$$(8)\quad a = \frac{v+1}{uT_o^v}$$

where T_o is already determined in the estimation of $h(t)$ and $g(t)$.

By meeting these two conditions, the values of parameters v and u of the $H(t)$ function are determined, since expressions (6) and (8) yield the implicit form

$$(9)\quad \frac{1}{v+1} = \left(\frac{t_o}{T_o}\right)^v$$

which has a unique solution for v and by substitution into (6) and (8) the value of u obtains.

Figure 3 shows the tabulation of the estimated parameters of the $h(t)$ and $H(t)$ functions as obtained from the ratings of two selected subjects for an assortment of dessert items. The last three columns are obtained from the first four columns, with the exception of T_o, which is the subjects' preferred time interval for the items and is part of the input data. It is noticeable that for most items the value of v is fairly close to one.

For reasons of analytical simplicity, the $H(t)$ function will be used as the analytical model of preference-time relations in the following parts of this paper. One can, however, further simplify the function by assuming that $v \approx 1$ for most items. This way a two-parameter approximation of the preference-time function obtains in the form

$$(10)\quad p(t) = a - \frac{1}{ut}$$

where parameters a and u are to be estimated from data. It is of some practical significance that conventional food preference questionnaires are sufficient to estimate these two parameters. In most of these questionnaires the subjects are asked to indicate how frequently they want to eat a given item, and they also have to rate their preferences for the item on some hedonic scale. With the assumption that their preference rating is conditioned by the estimated time interval T_o, corresponding to their frequency rating, the preference ratings can be regarded as direct estimates of $p(T_o)$, i.e., a point on the preference-time function. This assumption, of course, can be made operational by the appropriate phrasing of the questions.

With the estimated values of $p(T_o)$ and T_o available, (10) is one of the conditions, and

$$(11)\quad \frac{d}{dt}\,\frac{p(t)}{t} = -a + \frac{2}{ut} = 0$$

is the other condition that the parameters a and u must satisfy. Both conditions are satisfied if

$$(12)\quad a = 2p(T_o)\ ; \quad u = 2/aT_o$$

The analytical properties of this simplified preference-time function imply that the estimated preference at $t = \infty$ is twice as much as the preference at T_o time interval. It is interesting that the observed preference-time data (Figure 3) are not too far from satisfying this property, since the estimated value of the parameter v was fairly close to one.

In the previous part the preference-time function was introduced as a measure of satisfaction if a fixed portion of a food is consumed by an individual at identical time intervals of length t. Here we consider a particular food item (or menu item) denoted by the subscript j, and will utilize the $H(t)$ and $G(t)$ functions to get an expression for the measure of satisfaction over the whole planning horizon of a menu plan.

Let N be the total number of days included in the menu plan. With a serving time interval of t days, the number of times menu item j is offered is denoted by x_j. With the standardized portion size for which the preference-time functions are evaluated, the following identity holds

$$(13)\quad x_j = N/t \quad .$$

Consequently, the preference at each consumption as a function of the frequency, the number of times item j is offered on the menu plan, is

$$(14)\quad H(x_j) = a_j - (1/b_j)(x_j/N)^{v_j} \ .$$

The total preference derived by offering menu item j x_j times is obtained by multiplying $H(x_j)$ by x_j to yield $G(x_j)$.

$$(15)\quad G(x_j) = a_j x_j - (1/b_j)(x_j/N)^{v_j} \cdot x_j$$

or more simply

$$(16)\quad G(x_j) = a_j x_j - b_j' \cdot x_j^{v_j+1}$$

where

$$b_j' = (1/b_j) \cdot (1/N)^{v_j}$$

It is readily seen that $G(x_j)$ is a unimodal function of x_j, implying that it has a unique maximum at some value of x_j. This implies that individuals tend to pace their consumption of food j on the time scale such that the preferred number of times in the cycle length N is given by the maximum of the $G(x_j)$ function. In other words, for menu item j , $G(x_j)$ is the preference objective function to be maximized by one individual. $G(x_j)$ is a quadratic function if $v_j = 1$.

Until now, $G(x_j)$ was tacitly assumed to be the preference function of a given individual for food j. It is very likely that different individuals may have nonidentical preferences for the same set of foods. Consequently, the notation $G_i(x_j)$ will be introduced as the preference function of the i-th individual of a given population for food item j.

In particular

$$(17) \quad G_i(x_j) = a_{ij}x_j - b_{ij}x_j^{v_{ij}+1}$$

will replace the notation used in (16). The parameters a_{ij}, b_{ij} and v_{ij} can be established from questionnaires by the earlier described methods. Clearly, the $G_i(x_j)$ functions are concave for each individual i , so one can express the preference function of a set of individuals, M , for item j as follows:

$$(18) \quad G_M(x_j) = \sum_{i\in M} G_i(x_j)$$

where $G_M(x_j)$ will have a unique maximum at some value of x_j, since the sum of concave functions is also a concave function. Thus, one can say that for a given population the preference realized from food item j is a maximum for x_j is equal to x_j^o, where

$$(19) \quad G_M(x_j^o) = \max_{x_j} \sum_{i\in M} G_i(x_j) \ .$$

This is not to say, however, that the maximum of $G_M(x_j)$ functions of individuals in the population, especially if the population is very heterogeneous with respect to their preferences for foods. This problem can be resolved by partitioning the set of individuals into subsets, clusters such that the within cluster homogeneity of individuals is maximum with respect to the set of foods under consideration. The smaller the clusters are, the more clusters are needed, and the cluster maximum $G_M(x_j)$ will be closer and closer to the maximums of the $G_i(x_j)$ functions within the clusters. It suffices to say here that for any number of desired partitionings of set M , techniques of cluster analysis are available to find the most homogeneous set membership of clusters and thus the corresponding values of the $G_M(x_j)$ functions for any set of food items.

The function $G_M(x_j)$, as defined by (18), needs 3xM parameters for its evaluation. Parameter reduction can be performed on $G_M(x_j)$, too.

$$G_M(x_j) = \sum_{i\in M} (a_{ij}x_j - b_{ij}x_j^{v_{ij}+1}$$

$$= x_j[\sum_{i\in M} a_{ij} - \sum_{i\in M} b_{ij}x_j^{v_j}]$$

By noting that

$$x_j^{v_j} = \sum_{n=0}^{\infty} (v_j \ln x_j)^n/n!$$

the expression for $G_M(x_j)$ can be written as

$$G_M(x_j) = x_j[\sum_i a_{ij} - \sum_i b_{ij} \sum_n (v_j \ln x_j)^n/n!$$

$$= x_j[\sum_i a_{ij} - \sum_i b_{ij} - \ln x_j \sum_i b_{ij}v_j$$

$$- \frac{(\ln x_j)^2}{2!} \sum_i b_{ij}v_j^2$$

$$- \frac{\ln (x_j)^3}{3!} \sum_i b_{ij}v_j^3 - \cdots]$$

By denoting $w_n = \sum_i (b_{ij}v_j^n/n!)$

$$\bar{a}_j = \sum_i (a_{ij}-b_{ij})$$

the above expression reduces to

$$(20) \quad G_M(x_j) = x_j[\bar{a}_j - \sum_{n=1}^{\infty} (\ln x_j)^n \cdot w_n]$$

We note that the summation in the above expression forms a monotonically decreasing convergent series, and hence an arbitrarily accurate approximation to the $G_M(x_j)$ function can be obtained by retaining a finite number of terms in the summation. Although no tests have been performed to observe how fast the series converges, it seems that for large M , much fewer than 3xM parameters need to be used to obtain an accurate approximation to the function. It is realized that computing w_n initially uses all the original parameters. However, this need be done just once, and it can be done before the parameters are actually put to use in the nonlinear programming model described in the next section.

Expression (16) for $G(x_j)$ was obtained from (5), which was obtained from (3) to effect an economy in the number of parameters in the function. One can, of course, obtain the $G(x_j)$ function directly from (3), using all of the four parameters in it. The resulting function is

$$(21) \quad G(x_j) = ax_j - \frac{be^{-c'/x_j}}{1-e^{-r'/x_j}}$$

where c'=Nc and r'=Nr. Figure 4 displays the form of the $G(x_j)$ functions as obtained from a Skylab astronaut for two entrees.

III NONLINEAR PROGRAMMING SOLUTION TO THE MENU PLANNING PROBLEM

The fundamental problem of menu planning is to define which menu items and how many times should appear in a given time period - called a cycle - on the menu. According to the definition adopted here, menu planning is a decision problem which concentrates on the whole cycle, and attempts to find an optimum plan in terms of finding optimum frequencies of the items under consideration. In contrast, menu scheduling is defined as a problem of deciding which item should appear in which meal and day.

Scheduling and planning, of course, are intimately related in the sense that the time aggregate of the menu schedule for a cycle is the menu plan. In turn, a properly structured menu plan can be partitioned, i.e., scheduled into a sequence of menus. The presentation of the material will be based on this latter principle. For the sake of conceptual clarity, nonselective menus will be considered first.

It is assumed that a set of n menu items is subject to menu planning decision such that for each item j, optimum quantities x_j, $(j=1,2,\cdots,n)$ should be determined for a given set of individuals and for a menu cycle of s days. In this context the meaning of x_j is the number of unit portions of menu item j to be allocated on the menu during s days. It is assumed further that the set of n menu items can be partitioned into K subsets according to the course structure, such as entrees, starch, vegetable, etc. of the meals for a day. Let w_k $(k=1,2,\cdots,K)$ be the relative weight of course k in the total preference of the meals. With these notations, the total preference of a set of individuals M for n menu items can be expressed as follows:

$$(22) \quad G(X) = \sum_{k=1}^{K} w_k \sum_{j=n_{k-1}+1}^{n_k} G_M(x_j)$$

where $n_o=0$, $n_K=n$, $(n_k - n_{k-1})$ is the number of menu items in course k, X is the n-vector of x_j components, and

$$(23) \quad G_M(x_j) = \sum_{i\varepsilon M} a_{ij}x_j - b'_{ij}x_j^{v_{ij}+1}$$

as in (18), or as defined in (20) to achieve parameter economy. Consequently, G(X) is a weighted additive function of nonlinear expressions which all depend on the aggregate preference-quantity functions of a population for each of the n items involved. Inasmuch as the objective of menu planning is to select menu items for a cycle of s days which will be most preferred, this objective can be reached by finding the maximum of the G(X) function. To insure, however, that the resulting vector X is appropriate for the purposes of scheduling as well as from the point of view of other, such as budgetary, nutritional and compatibility considerations, only the constrained maximization of G(X) will provide, in general, acceptable results. This leads to a nonlinear programming formulation of menu planning.

Accordingly, menu planning with preference maximization objective can be formulated as a nonlinear program problem stated as follows:

$$(24) \quad \max.\ G(X) \quad \text{s.t.}\ c^TX \leq c_o$$
$$AX \geq B,\ MX \leq S,\ RX \leq D$$

where

$G(X)$ is the nonlinear objective function identical to (20).

c^T is the n-vector of unit portion costs of menu items.

A is the mxn matrix of the nutrient composition of menu items, with a_{ij} element indicating the amount of nutrient i in one portion of menu item j.

B is the m-vector of the nutrient allowances for some reference person for s days.

M is a Kxn incidence matrix containing staggered rows of unit coefficients corresponding to the availability of the items for given courses.

S is a K-vector of components s or $2s$ for nonselective menus indicating the number of items needed for a course for a cycle of s days.

R is an Lxn matrix of coefficients for assorted attribute constraints, proportionality constraints, production constraints, etc. which define feasibility conditions for scheduling the vector X.

D is an L-vector defined by the constraints above.

X is the vector notation for the menu plan which is fully defined by the values of the components of X. If the j-th component of X in the solution is not zero, x_j represents the number of portions of menu item j to be allocated for s days.

The above definition of x_j and its role with respect to the feasibility of scheduling requires that all the components of X be integers. Strictly speaking, menu planning is a nonlinear and integer programming problem. Such problems are still considered intractable in theory. In practice, however, the problem is not too serious because of two favorable factors. First, s can be rather large. Sixty day or 90 day menu cycles are common, so the number of portions to be represented by the x_j components can be large integers where the effects of rounding are relatively minor. Second, the nonlinear programming problem posed in (24) is well suited for solution by piecewise linearization and convex separable programming techniques where the grid points of the linearized variables can be conveniently selected to coincide with unit portions. This way all the upper bounds of the auxiliary variables will correspond to integer values and experience with such upper bounded linear programming models for menu planning has shown that in such cases a sizeable majority of the bounds tend to bind, and thus most of the components of the X-vector will be integer valued.

It should be mentioned here that an alternate nonlinear programming formulation of menu planning can be derived from (24) and considered as useful for institutional feeding programs where the management objective is to maintain a given food preference level at minimum cost. This version of the problem is explicitely stated here for further reference:

$$(25)\quad \text{Min. } c^T X \quad \text{s.t. } G(X) \geq u_o$$

$$AX \geq B,\ MX \leq S,\ RX \leq D.$$

Here u_o is some minimal level of preference to be maintained while the rest of the notations mean the same as in expression (24). This structure is similar, but not superior, to the "best buy" linear programming models discussed in [5], where the nonlinear constraint is replaced by a set of upper bounds.

The nonlinear programming problem (24), after the addition of slack and surplus variables can be written in the form

$$(26)\quad \text{max. } G(X) = \sum_{k=1}^{K} w_k \sum_{j=n_{k-1}+1}^{n_k} G_M(x_j)$$

$$\text{s.t. } \sum_j q_{ij} x_j = d_i \quad i=1,2,\cdots, m+K+L$$

$$x_j \geq 0$$

The above problem, although it is nonlinear and large in size, (well over 50 constraints and 400 variables for food service establishments), is amenable to efficient solution techniques in existence [13]. The objective function of problem (26) is additively separable, and this makes the application of what Wolfe [19] has termed grid-linearization particularly efficient.

We recapitulate briefly the features of a grid-linearization algorithm for the solution of the nonlinear additively separable problem (26).

An initial set of grid points $\{x_{jt}\}$ is defined for each variable x_j, yielding the linearized program in the variables λ_{jt}.

$$(27)\quad \sum_{k=1}^{K} w_k \sum_{j=n_{k-1}+1}^{n_k} \sum_t \lambda_{jt}\, G_M(x_{jt})$$

$$\text{s.t. } \sum_j \sum_t q_{ij} x_{jt} \lambda_{jt} = d_i \quad i=1,2,\cdots, m+K+L$$

$$\sum_t \lambda_{jt} = 1 \qquad \text{for all } j$$

$$\lambda_{jt} \geq 0 \qquad \text{for all } j,\ t$$

If the initial number of grid points for each variable in problem (27) is large, an acceptable approximation of the nonlinear objective function may result. However, if fewer grid points are used, new grid points can be created in the framework of the solution algorithm.

If, at iteration r, the linearized program is solved, an optimal solution $\{\lambda^r_{jt}\}$ and simplex multipliers (Π^r, Π^r_o) are available. For any variable x_j, a new grid point is sought such that it produces the most negative reduced cost factor. This corresponds to solving the unconstrained problem

$$(28)\quad \text{min. } \sum_{i=1}^{m+K+L} \Pi^r_i\, q_{ij} x_j + \Pi^r_{oj} - G_M(x_j)$$

Note that if $G_M(x_j)$ is concave, the above function in x_j is convex and the minimum is unique. The minimum in (28) can be efficiently determined by any appropriate technique such as the Method of Golden Sections [18]. The new grid point thus created can be added to the current set, and a new iteration begins.

Earlier convergence proofs [10] required that all columns be retained from iteration to iteration. However, recently Murphy [14] has developed some column dropping procedures, although no experience of the efficacy of such procedures is cited.

It may be mentioned that the constraints in problem (27) of the form $\sum_t \lambda_{jt} = 1$ can be handled as generalized upper bounds. Moreover, the columns generated can be implicitly stored.

IV DISCUSSION

An optimum integer vector X from solving either (24) or (25) produces only the menu plan which is to be scheduled by some other method. It is important to realize that the plan X already satisfies the most significant conditions pertaining to population preferences, total cost, nutrition, and other general aspects of feasibility. The only remaining objective of scheduling is to assure compatibility among the menu items within meals and between meals. Compatibility is a property of interactions between items, and not necessarily a property of the items per se. Thus, in scheduling we deal with acceptability aspects of combinations of items, which was not directly considered in the objective function. The model proposed thus separates the criteria of menu planning and scheduling into two distinct optimization processes for technical reasons. In the planning phase, food preferences are maximized as separable functions by the powerful method of nonlinear programming. In the scheduling phase, compatibility is to be maximized by techniques still in the exploratory stages. One obvious possibility is the arrangement of items by manual methods. More exact approaches are conceivable by algorithms based on multidimensional scaling and graph theoretical methods presently under investigation [4,12].

For populations which are heterogeneous with respect to their preferences for a given set of menu items, nonselective menus cannot be optimal [7]. An improvement on the optimality can be effected by partitioning the set of individuals M into subsets of individuals M_i, $(i=1,2,\cdots, t)$ where $\bigcup_{i=1}^{t} M_i = M$, and each of the t subsets will be more homogeneous with respect to preferences than the set M if the partitioning is done by cluster analytic methods [12]. In this case the $G_{M_i}(X^i)$ function for cluster M_i will have its optimum x^i_j values closer to the individuals' preference-quantity function optimums than is possible for set M. This is simply the theoretical interpretation of the rationale of offering selective

114

menus. In short, each cluster corresponds to a potential different optimum menu plan, hence a potential need for a choice on the menu which is expected to be exercised by the individuals in the particular cluster. By increasing the value of t, more homogeneous clusters can be created, more individual preferences will coincide with the respective $G_{M_i}(X^i)$ optimums, but also more choices will be needed and more kinds of items are to be prepared for each meal. It is conceivable that t, i.e., the potential number of distinct choices, has some practical bound, and there is evidence [7] that the greatest improvements in preferences occur at low values of t, such as $t=2$ or $t=3$.

The menu planning problem for a heterogeneous population is therefore pictured as the (joint) solution of t nonlinear programming problems with objective functions $G_{M_i}(X^i)$ corresponding to clusters M_i, $(i=1,2,\cdots,t)^i$ yielding optimum nonidentical menu plans X^i for each cluster. This is to say that the planning problem associated with selective menus is not particularly different from the one described earlier. The scheduling problem becomes, however, somewhat complicated.

Let us denote the cardinality of cluster M_i by m_i, and the cardinality of M by m_c where $m_c = \sum_{i=1}^{t} m_i$. The consistency between menu plans X^1, $X^2, \cdots, X^t$ and the corresponding selective menu schedule requires that if item j is represented in the plans in quantities $x_j^1, x_j^2, \cdots, x_j^t$, then the time averaged marginal probabilities of choosing these items from the schedule in s days must be equal to the marginal probability of choosing item x_j from other items in the given course by the total population in s days, which is equal - on the basis of cluster preferences - to

$$p_j = \frac{1}{sm_c} \sum_{i=1}^{t} m_i x_j^i \qquad (29)$$

It is a difficult and thus far unresolved problem to schedule selective menus which satisfy this criterion, because compatibility between menu items affects the joint probabilities of selections. Even if condition (29) is satisfied, the freedom of choice provided by selective menus introduces random variables in the food service system, and necessitates redefining the food cost, nutritional, and other constraints in probabilistic terms.

One can, of course, avoid some of these problems by scheduling each of the X^i option plans as nonselective menus, offering selectivity only among the menus, but not the items, and hoping that on the average, population cluster M_i will prefer to select the corresponding X_i menus from the schedule. Obviously, more research is needed on these points.

V COMPUTATIONAL RESULTS

The grid-linearizing algorithm to solve the preference-maximizing menu-planning problem was coded in FORTRAN for the CDC 6600. The program requires approximately 25000 words of active storage to solve 30-constraint, 320-variable problems, where the number of variables refers to the number before linearization. No passive storage is used. Approximately 7000 words of storage are problem-size-independent. Since integrality of the solution is desired, only integer grid points are used, and the program has the capacity to enforce different upper bounds on the variables.

Twenty-constraint, 220-variable problems require approximately 20 CPU seconds, and 20-constraint, 120-variable problems take less than 10 CPU seconds on the CDC 6600. Furthermore, if a "good" starting solution is used, the solution times decrease markedly.

Portions of the solutions to two sample problems are presented below. The nutrient and cost attributes are from a U.S. Army Master Recipe File. The problems consist of determining serving frequencies of menu items for a 42-day evening meal cycle. Although six-course meals were planned, the display contains frequencies of only the entrees under raw food cost budget limits of $42.00 and $35.00 for all the courses.

NO.	ITEM NAME	SERVING FREQUENCY (per 42 days) Budget=$35	Budget=$42
1	ROAST BEEF	3.00	4.00
3	GRILLED BEEF STEAK	3.00	5.70
6	SWISS STEAK W/BROWN GRAVY	1.00	2.00
8	MEAT LOAF	2.00	2.00
9	GRILLED SALISBURY STEAK	3.00	2.00
11	SWEDISH MEATBALLS	2.00	1.00
15	BAKED HAM	1.00	1.00
22	GRILLED SAUSAGE PATTIES	1.00	0
23	BARBECUED RIBS	.26	1.00
24	ROAST VEAL	1.00	1.00
26	BAKED CHICKEN	2.00	2.00
27	FRIED CHICKEN	4.00	4.00
28	ROAST TURKEY	1.00	1.00
29	HOT TURKEY SANDWICH	4.00	3.00
30	FRIED FISH	1.00	.30
36	FRENCH FRIED SHRIMP	2.00	2.00
37	SEAFOOD PLATTER	2.00	2.00
40	SPAGHETTI W/MEATBALLS	4.00	4.00
42	BEEF STEW	1.00	1.00
44	CHILI CON CARNE W/BEANS	3.74	3.00
	PREFERENCE	85.24	86.69

In both solutions, the use of integer grid points helps make the solution almost integer, with only two fractional values for the variables. We are currently using a search procedure to round the solution obtained from the nonlinear program by permitting fractional variables to change only to the next lower or next higher integers, and nonfractional variables to change by at most one. Other schemes such as in [1] are also possible.

Even with a 42-day budget reduction of 16%, the solution is able to exploit item substitution to yield less than a 2% reduction in acceptability. Unlike conventional trial-and-error menu planning procedures, this method prevents over-reaction to budgetary and price fluctuations.

An apparent shortcoming of the procedure is that acceptability and schedulability of the item frequencies obtained from the mathematical program depends on putting together day-to-day combinations of items that are compatible. The nonlinear program bypasses the issue of compatibility. However, blatant compatibility effects can be incorporated in the form of constraints. For example, if a sandwich appears on the menu, there may be no need to serve bread in addition. This situation can be handled via the constraint

$$\sum_{j\in S} x_j + \sum_{j\in B} x_j = N$$

where N is the number of days in the cycle, B is the index set of breads, and S is the index set of items that preclude bread from appearing on the menu. We enforce such "exclusion" constraints for several courses. Another type of constraint is the "inclusion" constraint, which enforces the appearance of one item if another appears. Thus Applesauce can be forced to appear as often as Pork Chops. In our experience, once the constraint set considers schedulability, the compatibility problem becomes easy to handle.

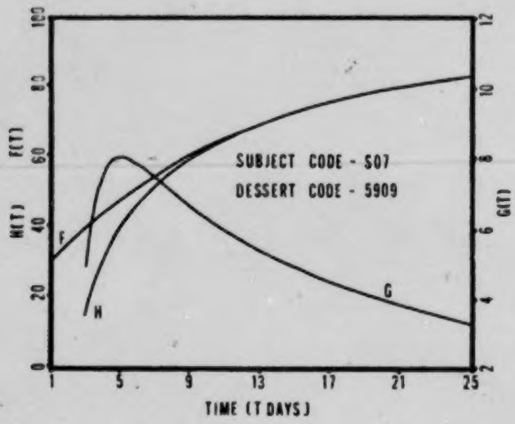


Figure 2. The shape of the f(t), h(t) and g(t) functions for parameter values of a=90.67, b=67.14, c=0.0884 and r=0.3760.
Source: Reference [6].

Subject/Item Codes	a	b	c	r	T_o	v	1/u
S06/5909	20.00	10.34	0.0336	1.3808	1.0	1.1450	0.2074
S06/6011	25.00	11.61	0.0374	0.2817	4.0	1.0099	0.0200
S06/5037	40.00	15.17	0.0834	0.1480	5.0	0.9590	0.0104
S06/5111	30.00	9.12	0.0859	0.0718	7.0	1.0631	0.00869
S07/5082	80.00	46.87	0.0628	0.1848	7.0	0.9089	0.00407
S07/5909	90.00	54.64	0.0624	0.2986	5.0	0.9189	0.00486
S07/5107	100.00	45.78	0.0895	0.3442	3.0	0.9288	0.00695
S07/5013	60.50	21.27	0.0451	0.0114	30.0	1.4754	0.000271

Figure 3. Estimated parameters of two analytical models of the preference-time function for foods.
Source: Reference [6].

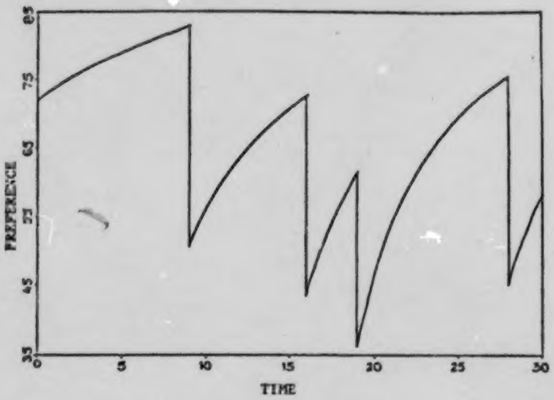


Figure 1. The change of preference over time when the item is consumed on days 9, 16, 19 and 28, the item having been consumed 9 days prior to day 1. The parameters of the preference-time function used for this plot are: a=100, b=40, c=0.05, r=0.4.
Source: Reference [7].

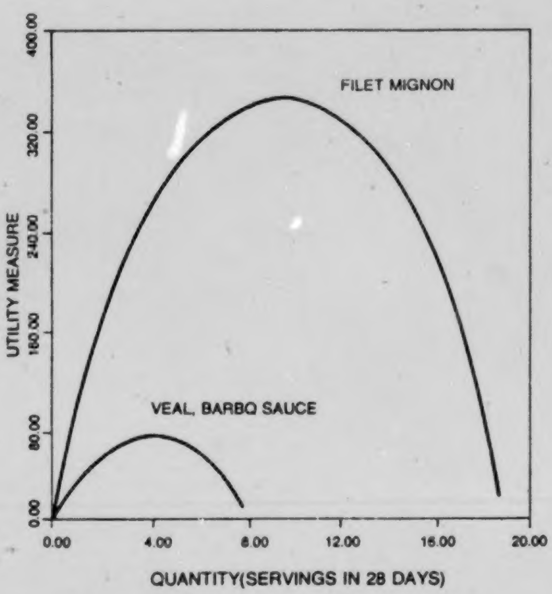


Figure 4. Unconstrained preference-quantity functions of an astronaut for two entrees.

REFERENCES

[1] Armstrong, R.D., J.L. Balintfy, P. Sinha, "A Multistage Scheduling Algorithm for Preference Maximized Selective and Nonselective Menus," ONR Technical Report No. 7, School of Business Administration, University of Massachusetts, July 1973.

[2] Balintfy, J.L., "Menu Planning by Computer," *The Communications of ACM*, Vol. 7, April 1964, pp.255-259.

[3] ______, "A Mathematical Programming System for Food Management Applications," *Interfaces*, Vol. 6, No. 1, Pt. 2 (1975) pp.13-31.

[4] ______, "Large-Scale Programming Properties of Menu Planning and Scheduling," in *Optimization Methods for Resource Allocation*, ed. R.W. Cottle and J. Karp, English Universities Press Ltd, London, 1976, pp.81-98.

[5] ______, J. Neter, W. Wasserman, "An Experimental Comparison Between Fixed Weight and Linear Programming Food Price Indexes," *J. Amer. Stat. Ass.*, Vol. 65, No. 329, March 1970, pp.49-60.

[6] ______, W.J. Duffy, P. Sinha, "Modeling Food Preferences Over Time," *Operations Research*, Vol. 22, No. 4 (1974) pp.711-727.

[7] ______, P. Sinha, "Computational Principles of Choicegroup Generation for Selective Menus," ONR Technical Report No. 5, School of Business Administration, University of Massachusetts, October 1972.

[8] Benson, P.H., "Psychometric Approach to Predicting Consumer Behavior," *Personnel Psychology*, Vol. 13, No. 1 (1960)pp.71-80.

[9] Bowman, J.D., E.M. Brennan, "Computer Assisted Menu Planning Provides Control of Food Service," *Hospitals*, Vol. 43, No. 103, August 1969, pp.107-113.

[10] Dantzig, G.B., *Linear Programming and Extensions*, Princeton University Press, 1963.

[11] Gelpi, M.J., J.L. Balintfy, L.C. Dennis, I.K. Findorff, "Integrated Nutrition and Food Cost Control by Computer," *J. Amer. Diet. Ass.*, Vol. 61, No. 6, December 1972, pp.637-646.

[12] Green, P.E., F.J. Carmone, *Multidimensional Scaling and Related Techniques in Marketing Analysis*, Allyn and Bacon, Boston 1970.

[13] Lasdon, L.S., *Optimization Theory for Large Systems*, The Macmillan Company, New York 1970.

[14] Murphy, F.H., "Column Dropping Procedures for the Generalized Programming Algorithm," *Management Science*, Vol. 19, No. 11, July 1973, pp.1310-1321.

[15] Peterkin, B., USDA-ARS. Personal Communication.

[16] Smith, Victor E., *Electronic Computation of Human Diets*, Michigan State University Business Studies, East Lansing 1963.

[17] Stigler, G.J., "The Cost of Subsistence," *J. Farm Econ.*, Vol. 27, No. 2, May 1945, pp.303-314.

[18] Wolfe, P., "Foundations of Nonlinear Programming," in *Nonlinear Programming*, J. Abadie, ed., John Wiley & Sons, Inc., New York 1967.

ON THE ANALYSIS AND COMPARISON OF MATHEMATICAL PROGRAMMING ALGORITHMS AND SOFTWARE

Ron S. Dembo
Yale University

John M. Mulvey
Harvard University

Abstract

Although mathematical programming algorithms and related computer software have been in existence for over twenty-five years, and new methods are being invented, revised and implemented at a rapid pace, there have been few (if any) concrete suggestions for conducting experiments to evaluate competing numerical techniques. Yet the computer-operations research folklore abounds with information about the reputed efficiencies of various programs. In an initial attempt at improving this condition, we analyze the problem from a statistical point of view. The approach is contingent on the existence of a well-defined population of test problems from which a statistical sampling is carried out. By measuring the relative performance of various codes on the sample problems, predictions can be made as to their relative performance on the population of problems under consideration. Furthermore, the significance of these predictions can be measured in a rigorous way using standard statistical procedures. As a demonstration of our methodology we consider in detail a comparison of a primal-simplex network code with one based on the out-of-kilter method. We show how one may define, in a precise manner, a population of test problems and what conclusions may be drawn from a simple random sampling procedure.

I. Introduction

Since the inception of the simplex method for linear programming thirty years ago and the simultaneous development of computers, researchers have been concerned with analyses and comparisons of mathematical programming techniques. At first, variants of the simplex method such as the revised simplex method, and product form of the inverse, were proposed as alternatives to the original simplex design and later as internal routines for a variety of nonlinear and combinatorial programming techniques. One of the first empirical analysis of these proposals was by Wolfe and Cutler [1963]. A plethora of computational studies have followed (cf. Kuhn and Quandt [1963], Florian and Klein [1970], Gilsinn and Witzgall [1973], Srinivasan and Thompson [1973], Zanakis [1973] and Barr et al [1974]). Paralleling these studies have been a host of informal unpublished experiences from which a substantial folklore about the relative effectiveness of various techniques has arisen; despite these studies, there is still little agreement today. Since the out-of-kilter network algorithm (a primal-dual approach) was published in Ford and Fulkerson [1962], for example, a debate has raged over the relative superiority of this method versus the network-specialized primal simplex methods (see Dantzig [1963], Aashtiani [1976], Klingman et al [1974], and Hatch [1975]).

There are several underlying causes for this lack of agreement: (1) the absence of a graded set of standard test problems, (2) uncertainties about the efficiencies of different computers, (3) incomplete descriptions of the experimental design variables when reporting computational experience, and (4) a lack of guidelines for performing computational experiments.

In other areas of mathematical programming such as nonlinear programming, attempts have been made to compare algorithms, see for example Colville [1970], Dembo [1975] and Rijckaert [1975]. Here, it is much more difficult to come to any firm conclusions regarding the relative performance of the particular codes in question than in the case of (say) network algorithms. Factors such as internal tolerance settings, accuracy of the solution and whether or not a code actually computed a Kuhn-Tucker point, all play a crucial role in evaluating the behavior of a coded alborithm.

There is no question as to the need for evaluating the relative performance of coded algorithms. From a theoretical point of view, algorithms are often evaluated on a "worst case" basis. This type of analysis is often misleading in a coded form of the algorithm. For example, it is widely accepted in the mathematical programming literature that Kelly's cutting plane algorithm [1960] is not a good method for solving convex programming problems in the sense of convergence rates and numerical stability. It is probably because of this folklore that one hardly ever hears of nonlinear codes based on Kelly's cutting plane algorithm. It is quite conceivable, however, that such a method might provide a basis for a relatively efficient and robust code for certain useful classes of mathematical programming problems. This has actually been demonstrated recently for small- to medium-sized geometric programming problems. In two independently conducted comparative GP studies (Dembo [1975] Rijckaert [1976]), a cutting plane algorithm was shown to be one of the most efficient[1] and robust[2] codes tested.

One of the major obstacles in conducting computational analyses involves conceptual differences between mathematical algorithms, the computer software and the detailed, problem-specific tactics which occur when empirical results are collected. Figure 1 graphically depicts the situation.

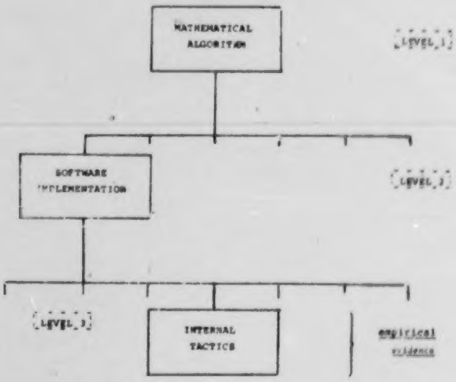


Figure 1
Conceptual Framework

Mathematical algorithms occur at the highest level of abstraction (level 1); following Zangwill [1969], we define algorithm as a sequence of point-to set mappings from which theoretical results can be derived - for example, infinite convergence properties, or algorithmic efficiency as defined by the number of iterations in a worse case analysis. Moving down to the next level of detail, we encounter computer software. For each mathematical algorithm (level 1) there are a host of computer software implementations (level 2) with widely varying degrees of effectiveness. These implementations range from undergraduate student LP projects to IBM's MPSX system. It is important to note that a basic, and distinguishing property of level 2 is the underlying information structure.

For each computer software implementation, there are usually many control settings which define internal tactics that the program utilizes in solving a particular problem or a problem-class. These control settings can result in drastic differences in efficiency, especially for large-scale mathematical programs. To illustrate the variation which can occur, a single assignment network with 10,000 nodes (constraints) and 30,000 arcs (variables) was solved (see Mulvey [1975]) with three different pivot strategies, that is, the procedure for selecting which eligible non-basic variable enters the basic at each pivot. The computational results are as follows:

Pivot Strategy	Seconds[1]	Pivots
(1)	3076	472,999
(2)	3028	28,113
(3)	971	136,204

A similar phenomena occurs in nonlinear programs where a small change in the tolerances causes large shifts in computational results (see for example Dembo [1975]).

Admittedly, the elements at each level (mathematical algorithm, software implementation, internal tactics) cannot be precisely defined and universally accepted. To some, a minor change in internal tactics is really a change in the fundamental mathematical algorithm. To avoid these difficult issues, we took a different approach for developing a framework with which we could analyze and compare mathematical algorithms and related software.

In the next section, we review the various types of test problems that may be used in numerical comparisons, and Section 3 provides the above mentioned framework by concentrating on well-defined collections of test problems. A statistical analysis is then undertaken in Section 4. In these experiments, it is not our primary intention to exhaustively test specific techniques, but to show how these comparisons can be conducted in light of a statistical analysis.

---

[1] In terms of standardized central processing time.

[2] In terms of the number of problems for which the code actually computed a correct solution.

---

[1] IBM 370/155 seconds, Fortran G compiler.

2. Test Problems

There are two main categories of problems that are currently used for reporting numerical results in the mathematical programming literature. Problems are either hand-selected or are randomly generated using a pseudo-random number generator. In both cases one encounters serious drawbacks when attempting a scientific analysis of computer codes used to solve a particular class of problems.

On the one hand, the behavior of computer codes on randomly generated problems does not in general reflect the behavior of these same codes on similar sized real world problems. This may be due in part to the fact that in real world problems there is usually some degree of correlation among variables. Correlation may be incorporated in randomly generated problems to accurately model real world behavior; however, both the nature of the application and the degree of correlation among variables would have to be known and thus the problem generator would have to vary from application to application.

On the other hand, if used for comparative purposes, computational results obtained from problems that have been selected from models of real-world situations make a statistical analysis of results questionable. In this case, conclusive statements regarding the performance of these codes can only be made for the particular problem set under consideration and any generalizations must be treated with suspicion.

The relative merits of the above categories of test problems are summarized below:

Hand-Selected Problems	Randomly Generated Problems
Usually representative or real-world behavior.	Usually not representative of real-world behavior.
Expensive to collect, document and send from one researcher to another.	Problem generators can be designed to be portable and machine independent.
Population of problems from which sample problems are drawn is not known. Thus, generalizations based on the sample are questionable.	Population of problems is known and can be controlled. If sampling method is known, generalizations based on sample statistics can be made with a known degree of certainty.

In this study we will restrict ourselves to pseudo-randomly generated problems. We have chosen to do so because inferences can be made about populations of test problems in a precise manner.

Since this is an initial attempt in a field that is relatively untouched, we have decided to restrict our attention to comparing the performance of two network codes on a well-defined population of assignment problems. Part of our aim will be to set out on defining a standard format for researchers to follow when reporting results on the behavior of coded algorithms.

The two codes considered in the next section are:

KILTER : A network code based on an out-of-kilter algorithm and written by Aashtiani [1976]

and

LPNET : A network code based on the primal simplex method and written by Mulvey [1975].

There are a number of reasons for starting our analysis with network codes as applied to the solution of assignment problems. First, network calculations involve manipulation of integers and therefore tolerances which are difficult to standardize and which play such an important role in comparing nonlinear programming algorithms, can be avoided. Secondly, a widely used pseudo-random problem generator, NETGEN (Klingman, Napier, Stutz [1974]), is available for constructing feasible network problems. Finally, as previously described, the recent literature on network codes has been filled with controversy as to whether either the out-of-kilter or the primal simplex method are the best approaches to solving assignment problems.

3. Notation and Methodology

The primary aim of computational comparisons is to make inferences about the relative behavior of the various algorithms under consideration. It is widely recognized that such comparisons cannot lead to any hard conclusions regarding algorithms themselves; rather, one can only derive information on the relative performance of the software implementations of these algorithms. What is not realized in most cases is that the problem of comparing software performance is a statistical one. Namely, the behavior of a number of computer codes on a specially selected set of problems is measured in terms of certain performance indicators and from this data inferences are made about the behavior of these codes on a larger class of problems. This is clearly a case of statistical sampling and in order for these inferences to have any firm basis, an experimental design should be carefully thought out with a view to the nature of the inferences that are to be made. Such a decision must

(i) identify the population from which sampling is to take place,

(ii) describe the statistical sampling method, and

(iii) state the hypotheses that are to be tested.

If the above three factors are carefully developed, then once the computer runs have been made and the appropriate variable measured, inferences can be drawn as to the relative behavior of these codes on the population of problems identified in (i)[*]. Furthermore, the significance of these inferences can be accurately measured using well known statistical techniques. In order to demonstrate the methodology, we perform, in detail, a statistical comparison of the network codes KILTER (Aashtiani [1976]) and LPNET (Mulvey [1976]).

In order to describe a general framework we need to define the following sets.

> *P is the set of all problems that the mathematical algorithm upon which the code is based, is theoretically capable of solving.*

For example, P(KILTER) is the collection of all transshipment problems solvable by the out-of-kilter algorithm and P(LPNET) is the set of all transshipment problems solvable by network specialized primal simplex algorithms.

> *P_c is the set of all problems for which the particular code was designed.*

For example, P_c(KILTER) is the set of all transshipment problems with less than 2000 arcs and 500 nodes and P_c(LPNET) is the set of all transshipment problems with less than 2000 arcs and 500 nodes.

> *P_t is the population of test problems and is a subset of $\bigcap_{i \in I} P_{ci}$, where I is the set of codes under investigation.*

In our case, P_t is taken to be the set of all feasible assignment problems as generated by NETGEN (Klingman, Napier Stutz [1974]), with the following characteristics:

- number of nodes between 200 and 500,
- number of arcs between 1000 and 200, and
- range of cost coefficients between a lower bound of 1 and an upper bound greater than 100 but not exceeding 5000.

Here, P_t was chosen because of the physical limitations of the computer used,[1] our self-imposed restriction that the runs should be carried out internally,[2] and the desire to compare these codes with respect to sparse, small-scale assignment problems. Table 1 below gives the details of the storage (high-speed memory) requirements and other important factors that are required for reproducability of our experiment.

Table 1

CODE SPECIFICATIONS

	P(KILTER)	**P(LPNET)**
Internal Memory Requirements	**37K Words**	**16K Words**
Language	**Standard Fortran**	**Standard Fortran**
Computer	**DEC 1070 (HBS[3])**	**DEC 1070 (HBS)**
Compiler	**Fortran-F40**	**Fortran-F40**
Precision	**Integer**	**Integer**

A typical sampling procedure would be the following. Draw a simple random sample, p, of n problems from the test problem population P_t. The size of n is chosen to be large enough so as to ensure, for example, that the distribution of sample means (of the particular performance measure, e.g., run time) is normal. Methods for choosing n are described in any elementary statistics text. A deciding factor in the choice of n might be a limit on the acceptable probability of rejecting an hypothesis that is actually true (Type I error).

In our case the simple random sample was chosen with the aid of NETGEN, a feasible pseudo-random network problem generator developed by Klingman, Napier and Stutz [1974][4]. This choice was primarily guided by a desire to make our experiment easily reproducable; NETGEN serves this purpose since it may be readily obtained from its authors and is already widely used. Details of our simple random sampling procedure are given below.

---

[*] The nature of the population and exactly what it represents is not addressed here and is a subject for future research.

[1] All testing was performed at Harvard University on the DEC 1070 computer with a maximum of 64K words of high-speed memory.

[2] Without resorting to auxiliary memory.

[3] Harvard Business School.

[4] The authors are willing to perform identical experiments with other generators provided the I/O formats conform to the SHARE convention and the generating program can be successfully compiled on the DEC 1070 computer (FORTRAN).

Procedure for Selecting a Simple Random Sample of Test Problems from the Population P_t

Step 1. Generate a random integer, d, in the range 100 to 5000. This fixes the range of cost coefficients to the range 1 to d.

Step 2. Generate a random integer, between 1000 and 2000. This fixes the number of arcs (variables) to n_1.

Step 3. Generate a random integer, m, between 200 and 500. This fixes the number of nodes (constraints).

Step 4. Given the problem parameters specified in Steps 1, 2 and 3 use NETGEN to generate a single feasible assignment problem with n_1 variables (Step 2) and m constraints (Step 3) and cost coefficients randomly selected from a uniform distribution in the range 1 to d.

Step 5. Repeat steps 1, 2, 3 and 4 until the desired sample size has been reached.

We should note here that we assume that the number of arcs and the number of nodes are uniformly distributed in the ranges 1000 to 2000 and 200 to 500 respectively.

In order to perform the statistical analysis in Section 4, we selected two independent simple random samples each containing 50 problems. The problem sets are all the necessary information required to reproduce them are given in Tables 2 and 3 below. The optimal solutions as well as the CPU run times required by LPNET and KILTER are given in Tables 4 and 5.

In the following section we perform a statistical analysis of the results.

Table 2

PROBLEM SPECIFICATIONS FOR FIRST FIFTY TEST CASES

OBS	# Nodes	# Arcs	Maximum Cost Coefficient	Density	Random Number Seed
1	284	1426	2176	.071	284
2	422	1376	4394	.031	422
3	466	1160	4930	.021	466
4	488	1776	4086	.030	488
5	234	1985	1041	.145	234
6	430	1447	2859	.031	430
7	474	1082	1136	.019	474
8	434	1217	539	.026	434
9	304	1628	2144	.070	304
10	486	1080	2081	.018	486
11	324	1669	4206	.064	324
12	462	1395	2553	.026	462
13	398	1353	2348	.934	398
14	456	1160	150	.022	456
15	268	1188	4421	.066	268
16	416	1813	689	.042	416
17	280	1579	3567	.081	280
18	282	1663	3185	.084	282
19	300	1775	1653	.079	300
20	396	1755	3987	.045	396
21	438	1111	4933	.023	438
22	296	1504	4444	.069	296
23	240	1722	2066	.120	240
24	496	1609	1830	.026	496
25	494	1897	2708	.031	494
26	268	1534	4235	.085	268
27	296	1870	3765	.085	296
28	340	1927	2603	.067	340
29	340	1731	3087	.060	340
30	348	1348	4653	.045	348
31	402	1880	4447	.047	402
32	242	1187	2594	.081	242
33	378	1672	1659	.047	378
34	216	1706	4520	.146	216
35	382	1497	2771	.041	382
36	290	1309	233	.062	290
37	494	1944	3865	.032	494
38	460	1097	3592	.021	460
39	482	1541	2837	.027	482
40	232	1260	3473	.094	232
41	434	1951	2522	.038	454
42	224	1493	3985	.119	224
43	386	1560	891	.042	386
44	364	1448	4823	.044	364
45	380	1853	4137	.051	380
46	242	1471	4809	.100	424
47	430	1836	1689	.040	430
48	202	1466	4176	.144	202
49	304	1157	1746	.050	304
50	278	1240	1980	.064	278

Table 3

PROBLEM SPECIFICATIONS FOR SECOND FIFTY CASES

OBS	# Nodes	# Arcs	Maximum Cost Coefficient	Density	Random Number Seed
51	442	1716	4200	.035	442
52	208	1611	2578	.149	208
53	420	1293	1063	.029	420
54	358	1428	4233	.065	358
55	266	1965	3811	.111	266
56	402	1673	1601	.041	402
57	466	1033	666	.019	466
58	402	1993	860	.049	402
59	342	1063	2493	.036	342
60	348	1598	2345	.053	348
61	462	1966	792	.037	462
62	362	1769	3152	.054	362
63	478	1800	3412	.032	478
64	216	1689	1176	.145	216
65	352	1734	4280	.053	352
66	294	1283	3546	.059	294
67	258	1479	1279	.089	258
68	234	1747	1244	.128	234
69	356	1469	1999	.046	356
70	292	1798	3520	.054	292
71	330	1180	1450	.043	330
72	274	1076	1769	.057	274
73	422	1464	2261	.033	422
74	452	1168	1704	.023	452
75	326	1530	3003	.058	326
76	236	1676	4062	.120	236
77	450	1267	2074	.025	450
78	356	1812	3906	.057	356
79	246	1450	122	.096	246
80	238	1225	2545	.087	238
81	356	1102	3719	.035	356
82	216	1205	1239	.103	216
83	488	1536	3201	.026	488
84	468	1863	128	.034	468
85	244	1119	1751	.075	244
86	454	1550	4512	.030	454
87	308	1537	4549	.065	308
88	486	1499	4375	.025	486
89	442	1698	1476	.035	442
90	232	1615	2596	.105	232
91	250	1307	3163	.084	250
92	228	1258	4336	.097	228
93	372	1363	3278	.039	372
94	256	1468	181	.090	256
95	292	1508	734	.071	292
96	496	1583	4176	.026	496
97	420	1114	3781	.025	420
98	398	1020	3388	.026	398
99	382	1207	3049	.033	382
100	424	1657	4445	.037	424

Table 4

RUN TIME RESULTS FOR FIRST 50 PROBLEMS
(DEC 1070 seconds)

OBS	KILTER	LPNET	Optimal Objective Function Value
1	10.700	3.900	54393
2	15.900	6.800	232241
3	15.300	7.100	354043
4	17.600	9.400	235820
5	11.700	4.600	17057
6	13.900	6.400	141765
7	7.200	4.500	87847
8	16.100	6.500	36739
9	11.600	4.100	59286
10	13.600	6.900	171024
11	16.300	5.200	132492
12	20.600	6.300	175529
13	1.400	5.700	117919
14	8.700	6.000	10228
15	8.100	3.100	128061
16	23.200	7.100	30992
17	12.100	3.900	89323
18	16.000	3.900	77548
19	13.000	5.100	43025
20	19.500	7.100	154219
21	10.200	4.200	329131
22	13.200	4.600	109068
23	12.300	3.100	37046
24	24.300	7.900	118056
25	21.700	9.200	143626
26	9.300	3.800	97023
27	17.700	5.100	90408
28	18.900	5.100	18911
29	13.400	5.000	94035
30	14.300	4.100	190304
31	21.200	8.000	175347
32	6.900	2.300	56866
33	15.300	7.700	67607
34	14.600	3.300	69186
35	16.100	4.900	108440
36	5.200	4.900	6944
37	25.200	7.500	213927
38	10.600	5.400	260746
39	19.300	7.100	160290
40	6.300	3.300	75499
41	27.100	8.800	123985
42	12.600	3.100	70750
43	18.700	6.400	38006
44	12.800	5.600	190868
45	21.800	5.500	158790
46	13.000	3.600	110732
47	22.700	7.300	81686
48	8.800	3.000	39916
49	10.200	4.100	64907
50	10.700	4.100	57996

Table 5

RUN TIME RESULTS FOR SECOND FIFTY PROBLEMS
(DEC 1070 seconds)

	LPNET Time	Optimal Objective Function Value
51	6.950	218462
52	3.940	37489
53	6.500	61535
54	4.260	168882
55	3.690	72642
56	7.890	67737
57	5.700	55139
58	7.250	33258
59	4.090	118324
60	6.060	86894
61	7.800	41309
62	6.470	103369
63	8.220	188851
64	4.170	17434
65	5.990	144286
66	3.390	99763
67	3.400	32124
68	4.000	23338
69	5.810	83422
70	5.040	85787
71	4.600	66249
72	3.470	57980
73	7.470	109052
74	4.670	115475
75	5.950	97911
76	4.260	75170
77	6.520	130922
78	7.000	131819
79	3.600	2806
80	3.810	56397
81	4.100	172230
82	2.960	23385
83	8.600	208613
84	8.680	6732
85	3.320	46686
86	8.710	253162
87	4.310	139037
88	7.860	30102
89	8.250	80863
90	3.500	52679
91	3.100	72179
92	3.640	89093
93	5.590	148160
94	3.310	4304
95	5.110	19390
96	7.200	263007
97	5.400	226245
98	4.570	202128
99	5.650	153775
100	7.140	199224

4. A Statistical Comparison of Assignment Codes

Summary statistics for the sample sets are given in Tables 6 and 7.

Table 6

SUMMARY STATISTICS FOR FIRST FIFTY PROBLEMS

	Min	Max	Mean	Std.Dev.
Nodes	202	496	360.04	89.98
Arcs	1080	1985	1526.96	267.5
Cost	150	4933	2944.36	1438
Density	.018	.146	.058	.0337

Table 7

SUMMARY STATISTICS FOR SECOND FIFTY PROBLEMS

	Min	Max	Mean	Std.Dev.
Nodes	208	496	350.2	88.56
Arcs	1080	1993	1479.3	264.7
Cost	122	4549	2584.06	1330
Density	.019	.149	.059	.0337

Notice the relatively large standard deviations for the number of nodes and arcs, and the cost range; this is due to the usage of a uniform density function in generating problems. Since the problems within p are "small-scale" examples, we wanted the characteristics of the problems within p to be selected with equal likelihood. For this reason a uniform density function was employed.

Turning to the computational results displayed in Table 4, it is interesting to observe that, although KILTER is fully 2.7 times slower than LPNET on the average, there is some variability. Specifically, KILTER is only 1.45 times slower than LPNET for problem #14, and a similar occurance is demonstrated for problem #36.

Table 8

SAMPLE CORRELATION COEFFICIENTS FOR PARAMETERS
(PROBLEMS 1-50)

	Nodes	Arcs	Cost	Density	Run-time Kilter	Run-Time LPNET
Nodes	1	-.0530	-.1516	-.9822	.0400	.0333
Arcs		1	.0378	.2897	.6713	.3621
Cost			1	.1575	.0485	-.1071
Density				1	-.1377	-.0680
Run-Time-Kilter					1	.7286
Run-Time-LPNET						1

It is interesting to observe that LPNET and KILTER are moderately correlated among themselves with a positive correlation coefficient of .7286, and that their relative dispersions (means/standard deviations) are approximately equal (see Table 9).

The next set of experiments tested LPNET on problems 51 through 100. Notice in Table 9 that the summary statistics, that is, the mean run time, standard deviation and $s/\bar{x}$ ratio for the experiment, are approximately[1] equal to the summary statistics for the first 50 problems. Since this set is independent of the first 50 problems, we are able to examine the distribution of sample means of size 50 for LPNET run time. Because of the relatively large sample size, 50, the distribution of sample means is normal, regardless of the nature of the distribution of individual run times. The sample mean is 5.459 which is a point estimate of the population mean run time. The estimated standard error of the mean is 0.249[2]. The small standard deviation is a result of the relatively large number of observations in the sample p. In a similar fashion, a point estimate of the mean run time for KILTER is 14.7 and an estimate of the standard error of the mean is 0.75.

Table 9

RUN TIME SUMMARY STATISTICS

	KILTER	LPNET
First 50 Problems	$\bar{x} = 14.698$ $s = 5.223$ $s/\bar{x} = .355$	$\bar{x} = 5.426$ $s = 1.750$ $s/\bar{x} = 0.323$
Second 50 Problems		$\bar{x} = 5.459$ $s = 1.743$ $s/\bar{x} = 0.319$

The distribution of sample means may also be used to generate confidence intervals on the mean CPU execution times for

---

[1] There are many useful tests that may be executed with data obtained from two independent simple random samples. For example, inferences made using the first sample may be checked using hypothesis testing based on statistics measured on the second sample.

[2] Standard error of the mean $\hat{\sigma} = s/\sqrt{n-1}$.

124

the population of problems, P_t, for both KILTER and LPNET. If we denote by μ_K, and μ_L the mean CPU time in seconds for KILTER and LPNET for the population P_t, we can construct the following 95% confidence intervals:

$$13.238 \leq \mu_K \leq 16.162$$
$$4.936 \leq \mu_L \leq 5.916$$
$$7.704 \leq \mu_K - \mu_L \leq 10.798$$

Since CPU times can be influenced by the number and type of other programs which were operating during execution, we carried out the following experiment to measure the extent of the variability which can occur in the DEC-1070 environment at Harvard. A single test problem (456 nodes, 1512 arcs, cost range [1,1414], seed 228) was solved by LPNET at seven randomly selected times during a one week period. The resulting CPU times are:

1) 8.400 seconds
2) 8.331 "
3) 8.933 " (≈ fully occupied)
4) 8.210 "
5) 7.966 "
6) 7.919 " (≈ empty)
7) 8.575 "

Thus from the above data we can expect the maximum error in measured CPU time to be of the order of 13%. It should be noted that these variations are caused by automatic "swapping" of jobs between central memory and auxiliary storage. This phenomenon occurs whenever a computer installation performs in a multiprogramming environment.

A linear regression analysis based on the first sample of 50 problems was undertaken to determine the overall combined effects of problem specifications and performance, as measured by CPU time. Several regression models were tried. The most appropriate linear regression equation for LPNET was found to be:

$$Y/\text{LPNET} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

where

Y/LPNET = run time for LPNET in seconds
x_1 = number of nodes
x_2 = number of arcs

and the estimated coefficients are given by

β_0 = -4.393 with a standard deviation of .72
β_1 = 0.01654 with a standard deviation of .00120
β_2 = 0.00255 with a standard deviation of .00038

The estimated standard deviation of the residual for the above regression equation was .72 and the coefficient of determination (sample R^2) was computed to be 0.845. The relatively* small standard deviations of the sample regression coefficients β_0, β_1 and β_2 indicate that the sample estimates β_0, β_1 and β_2 are significant. Another way of saying this would be that the CPU run time of LPNET depends to a significant extent on the number of nodes and number of arcs in the problem being solved. The high R^2 (.845) means that 84.5% of the variance can be explained by the size of the problem as measured by the number of nodes and arcs.

Although the relationships are approximately linear over this collection of problems, we do not expect that a strictly linear extrapolation can be made to larger problems. Caution should be taken not to use this regression equation to make predictions for populations other than the one considered in this study.

A second regression model fits KILTER's run time to the problem specifications with the resulting coefficients:

$$Y/\text{KILTER} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

where

Y/KILTER = run time for KILTER in seconds
x_1 = number of nodes
x_2 = number of arcs

and the estimated coefficients are given by

β_0 = -17.63 with a standard deviation of 2.63
β_1 = 0.03267 with a standard deviation of .00409
β_2 = 0.01347 with a standard deviation of .001376

The estimated standard deviation of the residual for the above regression equation was 2.601 and the coefficient of determination (R^2) was computed to be .767. This

---

*The quantities of interest are actually:

$$\beta_0/\sigma_{\beta_0} = \frac{4.393}{.72} = 6.1$$

$$\beta_1/\sigma_{\beta_1} = \frac{.01645}{0.0012} = 13.7$$

$$\beta_2/\sigma_{\beta_2} = \frac{0.00255}{0.00038} = 6.7$$

Since these numbers are much larger than 3 and the β_j are approximately normally distributed we conclude that β_0, β_1 and β_2 are all significantly greater than zero.

model is not as predictive as the previous, but R^2 remains relatively high; here, 76.7% of the variance can be explained by the number of nodes and arcs.

When an independent variable representing the cost range was added to this model the sample R^2 increased slightly to 77.7%. This insignificant improvement did not warrant inclusion of a third independent variable. Hence the simpler two variable model was selected.[1]

From these two regression models, we see that LPNET seems to depend to a greater degree on the number of nodes than the number of arcs in the network ($\beta_1 = .01645$, $\beta_2 = .00255$), whereas KILTER depends upon the number of nodes to a much lesser degree ($\beta_1 = .03267$, $\beta_2 = .01347$). Loosely speaking, this means that LPNET is more node-dependent than KILTER, while KILTER is more arc-dependent than LPNET. This result is intuitively appealing since the primal-simplex algorithm LPNET works with a basis spanning tree (m-nodes) which must be maintained -- thus the dependence on the number of nodes. The out-of-kilter algorithm, on the other hand, works primarily with arcs which are "out-of-kilter" -- thus the greater dependency on the number of arcs.

Besides being used for understanding the interdependence of problem characteristics and program performance, these regression analyses can be employed for forecasting CPU times for problems in the population P_t. This can be done as follows by making point estimates by substituting for x_1 and x_2 in the regression equations or by setting up confidence intervals for the predicted run times Y/LPNET and Y/KILTER.

For example, a point estimate of the LPNET run time for a problem with 456 nodes and 1160 arcs (see problem 14) would be:

Y/LPNET = 6.058 seconds

The actual run time for problem 14 was 6.000 seconds. Here the error in the point estimate is approximately 1%.

A 98% confidence interval for Y/LPNET would be:

$$\bar{Y}_{LPNET} - B_{.98/2}\,\hat{\sigma}_{\bar{Y}} \leq Y_{LPNET} \leq \bar{Y}_{LPNET} + B_{.98/2}\,\hat{\sigma}_{\bar{Y}}$$

where $\bar{Y}_{LPNET}$ is the point estimate run time, $\hat{\sigma}_{\bar{Y}}$ is the estimated residual standard deviation of $\bar{Y}$, and $B_{.98/2}$ is the standard normal deviate.

Hence, in our case with 98% certainty

$$6.058 - 2.303(.72) \leq Y_{LPNET} \leq 6.058 + 2.303(.72), \text{ or}$$

$$4.400 \leq Y_{LPNET} \leq 7.716$$

Thus we predict with 98% certainty that for problems with 456 nodes and 1160 arcs and any cost coefficients falling in the range defined by our population LPNET, will take no longer than 7.716 seconds and no less than 4.400 seconds to find an optimal solution.

Clearly, many other statistical tests could be carried out, depending upon what the specific objectives of the experiments were. Since our aim is to introduce a methodology rather than exhaustively compare KILTER and LPNET, we did not conduct further testing.

5. Conclusions

The statistical analysis in Section 4 shows that the code LPNET is clearly superior to the code KILTER for problems within the collection P_t. The conclusions are not unexpected since LPNET dominates KILTER. Nonetheless, we are able to measure precisely the extent of the superiority and propose confidence limits with a statistical framework. Despite this analysis, there are many unanswered questions about LPNET's superiority to problems outside of P_t but within collections P_c and P. It is an open question whether similar conclusions would be obtained if the objective of the comparative study was to evaluate how well the codes reoptimize, that is, how well the codes restart from a nearby basic feasible solution. Thus we caution the reader to treat these results as conclusive over set P_t, but not outside of this domain.

The primary purpose of this report was to develop an initial framwork for analyzing and comparing mathematical programming software. Obviously there is much unfinished work to be accomplished. A thorough study of problem generators as relating to real-world examples is an important next step. If the complexity of real-world problems could be better understood, it might be possible to design problem generators which more closely reflect the characteristics of realistic problems. The following idea which proposes a synthesis of the two categories of test problems described in Section 2 might be useful for building generators.

Start by choosing a representative test case t from the class of problems under study, for instance, a particular nonlinear programming design problem. Place perturbations on various parameters, such that the cost coefficients, and call the resulting population P_t. (It is left up the researcher to select the appropriate parameters and ranges.) From this population a sample p is drawn and inferences about P_t are made. Since the population P_t is defined to be within an ε-neighborhood of the original test case, conclusions about P_t, in some sense, reflect a sensitivity analysis for problem t. For instance, the standard deviation of run time measures how sensitive the performance of the code is to small changes in the original data. As a secondary consideration, it might be interesting to execute the programs from different starting points.

---

[1] In a similar analysis carried out on larger problems, the cost coefficient range became more important.

These concepts fall within the purview of experimental design. Since it is important to minimize the computer costs for testing, especially for large-scale examples, a sound experimental design can reduce the variances of the estimates and thereby result in lower computational costs. As an example, the simple random sampling procedure which was performed in this paper can be replaced by stratified sampling.

Another important area for future research lies in developing a clearer view of the relationships between the sets P_t, P_c and P. An estimate on the upper and lower bounds of CPU time for problems within P_t might lead to an accurate estimate for average CPU time for problems within P_c. The same might be said for P as well.

The ultimate benefit of the statistical framework is for assessing competing software codes or systems so that a systematic choice can be made as to the best technique for a particular user. Clearly this decision is a difficult multi-attributed problem and until these types of decisions can be handled, we must be content to list a profile of characteristics and performance measures for each code. From this profile, the user can select the appropriate technique for his or her particular needs.

Finally, we should mention that the working committee on algorithms (WCA) of the Mathematical Programming Society is actively engaged in computational-related research. (The authors are members of the committee.) The ambitious goals of the WCA are:

1) collect a graded set of test problems,
2) act as a focal point for knowledge of computer programs that are available for the same calculation,
3) recommending "best buys" where several techniques are available for the same calculations,
4) encouraging persons who distribute programs to meet certain standards of portability, testing, ease of use and documentation, and
5) define guidelines for conducting and reporting computational experiences.

One of the most important, and most difficult, goals is the last -- to define journalistic guidelines. The imposition of fair guidelines could markedly improve the state of computational work by providing better information to researchers and software users.

Acknowledgements

The authors would like to thank Norman Archer, George Wesolowsky and Stanley Zionts for their constructive criticism of an earlier version of this paper.

References

Aashtiani, H. "Solving Large-Scale Network Optimization Problems by the Out-of-Kilter Method," Master's thesis, MIT, 1976.

Aashtiani, H.A., and T.L. Magnanti, "Complementing Primal-Dual Network Flow Algorithms," Operations Research Center, MIT, OR 055-76, 1976.

Barr, R.S., F. Glover, and D. Klingman, "An Improved Version of the Out-of-Kilter Method and a Comparative Study of Computer Codes," Mathematical Programming, 7, 1, 1974.

Colville, A.R., "A Comparative Study of Non-Linear Program Codes," in Proc. Princeton Symposium on Math. Programming, Princeton University Press, 1970.

Dantzig, G.B., Linear Programming and Extensions, Princeton University Press, 1963.

Dembo, R., "Current State of the Art of Computer Codes for Geometric Programming," Presented at the ORSA/TIMS Chicago conference, May 1975. (To appear.)

Florian, M. and M. Klein, "An Experimental Evaluation of Some Methods of Solving Assignment Problems," Can. Op. Res. Soc. Journal, 8, 1970.

Ford, L.R. and D.R. Fulkerson, Flows in Networks, Princeton University Press, 1962.

Gilsinn, J. and C. Witzgall, "A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees," NBS Technical Notes 772, 1973.

Glover, F., D. Karney, and D. Klingman, "Implementation and Computational Study on Start Procedures and Basis Change Criteria for a Primal Network Code," Networks, 20, 1974.

Glover, F., D. Karney, and D. Klingman, and A. Napier, "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," Networks, 20, 5, 1974.

Hatch, R.S., "Bench Marks Comparing Transportation Codes Based on Primal Simplex and Primal-Dual Algorithms," Operations Research, 23, 6, 1975.

Kelly, J.E., "The Cutting-Plane Method for Solving Convex Programs," Journal of the Society for Industrial and Applied Mathematics, 8, 4, 1960.

Klingman, D., A. Napier, and G.T. Ross, "A Computational Study on the Effects of Problem Dimensions on Solution Time for Transportation Problems," University of Texas at Austin Research Report CS 135, 1973, to appear in JACM.

Klingman, D., A Napier, and J. Stutz, "NETGEN: A Program for Generating Large-Scale Assignment, Transportation, and Minimum Cost Flow Network Problems," Management Science, 20, 5, 1974.

Kuhn, H.W. and R.W. Quandt, "An Experimental Study of the Simplex Method," Proc. Symposia in Applied Mathematics, 15, 107-124, 1963.

Lee, S., "An Experimental Study of the Transportation Algorithms," Master's thesis, Graduate School of Business Administration, UCLA, 1968.

Mulvey, J.M., "Testing of a Large-Scale Network Optimization Program," Harvard University Working Paper, HBS 75-38, 1975.

Rijckaert, M.J. and X.M. Martens, "A Comparison of Generalized Geometric Programming Algorithms," Katholieke Universiteit Leuven, Report CE-RM-7503. 1975.

Smith, D.M. and W. Orchard-Hays, "Computational Efficiency in Product Form LP Codes," in R. Graves and P. Wolfe (eds.), Recent Advances in Mathematical Programming, McGraw-Hill Book Company, Inc., 1963.

Srinivasan, V. and G.L. Thompson, "Benefit-Cost Analysis of Coding Techniques for Primal Transportation Algorithms," JACM, 20, 1973.

Wolfe, P. and L. Cutler, "Experiments in Linear Programming," in R. Graves and P. Wolfe (eds.), Recent Advances in Mathematical Programming, McGraw-Hill Book Company, Inc., 1963.

Zanakis, S. "Experimental Comparison of Nonlinear Programming Algorithms in Deriving Maximum-Likelihood Estimates for the Three-Parameter Weibull Distribution," Ph.D. dissertation, Pennsylvania State University, 1973.

Zangwill, W.I., Nonlinear Programming: A Unified Approach, Prentice Hall, Inc., Englewood Cliffs, N.J., 1969.

This research project was partially funded by Associates of the Harvard Business School and the National Research Council of Canada, Grant # 214-1977-361.

128

THE EVALUATION OF UNCONSTRAINED OPTIMIZATION ROUTINES*

by

L. Nazareth
Applied Mathematics Division
Argonne National Laboratory
Argonne, IL

and

F. Schlick
University of Illinois
Champaign-Urbana, IL

Abstract

We discuss different approaches to evaluating optimization routines and describe a particular method which uses parameterized test problems. We illustrate this approach through a simple case study of three well known unconstrained optimization routines applied to three parameterized test problems. In particular we display our results as a set of graphs.

1. INTRODUCTION

Evaluating mathematical routines is a difficult task, and one that requires both qualitative and quantitative measures of performance. A fundamental requirement is that the testing environment simulate an actual environment of use since, if it did not, the evaluation would be valid, but in all likelihood, irrelevant. Furthermore, the overall quality of a code can only be gauged after investigating a broad range of issues, for example, efficiency, robustness, usability, usefulness of documentation, ability to fail gracefully in the presence of user abuse, rounding error difficulties or violation of underlying assumptions. A testing method usually concentrates on efficiency and robustness, evaluating these by exercising the code on a set of well chosen and hopefully realistic problems.

Our aim in this paper is to discuss some of the issues pertinent to the evaluation of optimization routines, and to describe an approach which employs parameterized test problems, first introduced for the purpose of evaluating routines for numerical quadrature by Lyness and Kaganove [1]. We describe in Sections 2 and 3 our particular motivation for and usage of such functions. To illustrate this approach three well known optimization routines were evaluated in a simple case study. In Section 4 and 5 we describe the routines and test problems, and present the results in graphical form. We believe that this case study gives some interesting examples of how unconstrained optimization routines behave when applied to parameterized test problems. The study is, however, very limited in scope, and we discuss some of these limitations in the concluding section.

2. DIFFERENT APPROACHES TO TESTING

To date the most common method of evaluating optimization routines has become known as 'battery' or 'simulation' testing. The first really comprehensive study along these lines is that of Hillstrom [2]. Battery testing has two basic components, namely, a set of test problems given by an objective function and a starting vector, and a number of measures of performance.

Test functions are chosen from the literature, or from real life applications, usually because they have some prominent feature, such as a curving valley possibly helical, a singular Hessian at the minimum, badly scaled variables or large dimensionality. The choice of starting points is, of course, crucial to performance. Many routines perform well from the standard starting points. However, as discussed in [2], the use of a number of widely dispersed starting points, reveals much about the strengths and weaknesses of a code, for example, how robust it is, the suitability of its convergence criteria, or the code's ability to handle long searches through non-quadratic regions.

A variety of different measures of performance have been used. The most common measure is the number of calls to the routine which develops information about the function, or the number of equivalent function evaluations; since the cost of gradient information is hardly ever the equivalent of n function evaluations (where n is the problem dimension), the appropriate weighting of gradients leads to the DCU or Horner unit scheme of Hillstrom [2]. Other measures include overhead and average rate of convergence.

Whilst battery testing yields very valuable information about the behaviour of optimization routines, it is nevertheless subject to limitations which often make a clear ranking of methods difficult to discern. First, it is difficult to know how much confidence should be attached to the measures of performance, since slight variation of starting point or geometry of test problem can lead to substantial variation in a performance measure. Second, when starting points are varied substantially, one can, in effect, get very different test problems. For example, Rosenbrock's function $f(x) = 100(x_1^2-x_2)^2 + (1-x_1)^2$ is more difficult to solve if started at the point (-2,2) than if started at (2,2); in the former case, a routine must follow a steep curving

---

*Work performed under the auspices of U.S. Energy Research and Development Administration.

valley, particularly taxing at the point (0,0). Thus, when evaluating a routine's ability to cope with a particular feature it may not be advisable to average a performance measure over widely differing starting points; however, if no averaging is done, one may be swamped with numbers.

In attempting to deal with some of these difficulties, we have utilized parameterized test problems, introduced originally into the evaluation of routines for numerical quadrature by Lyness and Kaganove [1]. We will not, however, use the Lyness-Kaganove term performance profile testing which involves a somewhat specific usage of parameterized test problems to rank "black-box" mathematical routines. Though we also want to compare routines, our usage of parameterized test problems is oriented toward the gathering of information on the performance of an algorithm, with a view to further development of its implementation. Thus our bias is toward the development of tools to aid the process of algorithm development rather than tools to aid mathematical software evaluation; though we are also concerned with the latter it is, in our opinion, a much harder problem. In the next section we discuss our method.

3. OUR APPROACH

Test problems are chosen in which a prominent feature can be varied by changing the value of a parameter λ occurring in the problems mathematical formulation. We have departed from the Lyness-Kaganove approach which is to parameterize a problem in such a way as to provide many different 'incarnations' of a test problem, all of approximately the same level of difficulty, by parameterizing e.g. the position of the peak of a unimodal function but not its shape. Varying our parameter alters significantly the overall difficulty of the problem. Thus one can test a routine's effectiveness with respect to a particular feature and study the routine's performance as this feature becomes more and more prominent. For example, the feature may be a steep curving valley, whose steepness or curvature increases with λ. At the same time measures of performance for values of λ in the neighborhood of any particular value, say $\bar{\lambda}$, will give an idea of the variability of the performance measures.

Starting points are, of course, critical to the minimization process and often bias the path of search. In addition to running each test using a conventional or fixed starting point, the initial points were varied using a random number generator. The motivation for this approach is twofold: 1) it was desired to obtain the "spread" between the maximum and minimum number of calls to FCN and thus determine how sensitive each routine is to a change in the starting point; and 2) by obtaining average values and using these for comparison purposes, it was hoped that the results would be less biased with respect to the starting points.

Clearly, an important consideration is how the points should be varied. We chose to vary starting points within a "box" surrounding the conventional starting point thus producing different initial points but the same general starting location with respect to the topography. A hypercube of dimension 0.1 units was used for all the tests in which the starting points were varied. Another important aspect is that the same starting points were used for each run (i.e., for each value of λ) by resetting the random number generator.

The statistics gathered for each test function include the following:

1. A table for each optimization routine using conventional starting points. The number of FCN calls, the number of iterations, the solution point, gradient, and function value for each λ are given.

2. A table for each optimization routine using the random starting points. The average, maximum and minimum number of FCN calls, and the average number of iterations for each λ are given.

3. Plots of the maximum, minimum, and average number of FCN calls versus λ for each routine used.

4. A superimposed graph of three plots (corresponding to the three routines) of average number of FCN calls versus λ.

The graphs were plotted using Fortran subroutines.

4. A CASE STUDY

In order to illustrate these ideas, a simple case study was carried out involving three routines and three parameterized test problems.

4.1 Routines Used

The following three unconstrained nonlinear optimization routines were used in this study.

1. CGMIN - a modularized version of the function minimization Harwell Library routine VA08A written by R. Fletcher [3] which uses the Fletcher-Reeves version of the conjugate gradient technique (see [7]).

2. BFGS - a modularized version of the Davidon-Fletcher-Powell quasi-Newton function minimization algorithm [4,5] with a BFGS (Broyden-Fletcher-Goldfarb-Shanno) [6] update to the Hessian (see [7]).

3. OCOPTR - a modularized implementation of C. Davidon's [8] optimally conditioned algorithm for calculating the minimum of a function of several variables.

A user-supplied subroutine FCN which evaluates the function value F and the components of the gradient G at the point $\underline{X}$ is needed by each of the above routines.

In addition, the user must supply the following information in the calling program to CGMIN:

1. the number of variables (i.e. the dimension of $\underline{X}$).
2. the initial estimate of the solution vector $\underline{X}$.
3. an estimate RFN of the expected reduction in F (used on the first iteration only).
4. the accuracy required in the solution, ε_1. The first condition for a normal return is that the differences between the components of two successive estimates of the solution are less than ε_1.
5. an additional accuracy requirement, ε_2. The second condition for a normal return is that the gradient norm is less than ε_2.
6. the limit on the number of calls to the function evaluation subroutine FCN. If the limit is exceeded before a minimum is attained, CGMIN will terminate abnormally with an appropriate error code.

For all of the runs, RFN was set equal to 1.0; ε_1 and ε_2 were set equal to 10^{-8}. For all three of the routines, the limit on the number of FCN calls was set at 200, 500, or 1000 depending on the test function.

The routine BFGS requires the following information from the calling program:

1. & 2. (same as for CGMIN).
3. the accuracy required in the solution, ε. A normal return occurs if the sum of the absolute values of the components of the solution difference and direction vectors are both not greater than ε.
4. an estimate of the minimum value of $F(X)$.
5. the limit on the number of calls to FCN.

In all cases, ε was set to 10^{-8} and the estimate of the minimum was set equal to 0.

The necessary parameters in the calling program to the optimization routine OCOPTR are:

1. & 2. (same as for CGMIN).
3. the limit on the number of FCN calls.
4. the accuracy sought ε. A normal return occurs if $g^T Hg < \varepsilon$, where H is the current approximation to the inverse hessian, and g is the gradient at the current iterate.
5. an estimate on the lower bound of the function. ε was set to 10^{-8}, and the lower bound estimate was set equal to 0.

4.2 Problems Selected

The following test functions were parameterized and implemented in the evaluation of the above algorithms (λ_0 represents the conventional or original value of the parameter):

1. Parameterized Rosenbrock test function ($\lambda_0 = 100$)

   a. $F = (1-X_1)^2 + \lambda(X_2-X_1^2)^2$

   b. $\dfrac{\partial F}{\partial X_1} = -2(1-X_1) - 4\lambda(X_2-X_1^2)^2$

   $\dfrac{\partial F}{\partial X_2} = 2\lambda(X_2-X_1^2)$

   c. conventional starting point is (-1.2,1.0).

   The function F has a global minimum at (1.0,1.0). The random starting points were generated over the component intervals $X_1(-1.3,-1.1)$, $X_2(0.9,1.1)$.

2. Powell Parameterized badly scaled function of two variables ($\lambda_0 = 10{,}000$)

   a. $f_1 = \lambda X_1 X_2 - 1, \quad f_2 = e^{-X_1} + e^{-X_2} - 1.0001$

   b. $F = f_1^2 + f_2^2$

   c. $\dfrac{\partial F}{\partial X_1} = 2f_1(\lambda X_2) - 2f_2 e^{-X_1}$

   $\dfrac{\partial F}{\partial X_2} = 2f_1(\lambda X_1) - 2f_2 e^{-X_2}$

   d. conventional starting point is (0.0,1.0).

   For $\lambda = 10^4$, the function F has global minima at $(1.098\times10^{-5}, 9.106)$ and $(9.106, 1.098\times10^{-5})$. The random starting points were generated over the component intervals $X_1(-0.1,0.1)$, $X_2(0.9,1.1)$.

3. Perturbed Quadratic Function

   a. $F = X_1^2 + 2X_2^2 + 3X_3^2 + 4X_4^2 + \lambda(X_1^4 + 3X_2^4 + 4X_3^4 + 6X_4^4)$

   b. $\dfrac{\partial F}{\partial X_1} = 2X_1 + 4\lambda X_1^3$

   $\dfrac{\partial F}{\partial X_2} = 4X_2 + 12\lambda X_2^3$

   $\dfrac{\partial F}{\partial X_3} = 6X_3 + 16\lambda X_3^3$

   $\dfrac{\partial F}{\partial X_4} = 8X_4 + 24\lambda X_4^3$

   c. fixed starting point is (10,20,30,40)

   The function F has a global minimum at (0,0,0,0). The random starting points were generated over the component intervals $X_1(9.9,10.1)$, $X_2(19.9,20.1)$, $X_3(29.9,30.1)$, $X_4(39.9,40.1)$.

All the tests reported in this paper were run in double precision (~16 significant digits) on an IBM 370/195. The numerical underflow and overflow, and the divide check interrupts were suppressed. The random starting points were obtained using the ANL Applied Mathematics Division library random number generator G552S.

5. TEST RESULTS

Graphical results described in Section 3 are given in Appendix 1 for the above problems. (Tabular results are not given because of space limitations.) Only the initial portions of the graphs are reproduced here, these being sufficient to illustrate the points raised during the discussion in Section 2.

a) Parameterized Rosenbrock's function
$F(X) = (1-X_1)^2 + \lambda(X_2-X_1^2)^2$

For $\lambda = 100$ this is a well known, difficult to minimize test function, which has a descending parabolic valley as its dominant feature. The starting point (-1.2,1.0) is chosen to bias the search down the valley. The parameter λ was varied from 20 to 1000 in steps of 20; increasing λ increased the difficulty of the problem by making the valley steeper. The limit on number of calls to FCN was set to 200.

Figure 1A shows quite clearly that the performance of CGMIN and BFGS are comparable, with OCOPTR performing substantially better. Figures 1B, 1C and 1D show the sensitivity to starting points of each routine. In Figure 1C note in particular how substantial the variation can be for certain values of λ, e.g. for $\lambda = 120$. Such a choice of λ in a battery test could suggest a very misleading ranking of the methods. Although we have not attempted to track down the reasons for the poor performance of the BFGS method in certain instances, such detective work would undoubtedly lead to improvements in the implementation.

b) Powell's Badly Scaled Function
$F(X) = (\lambda X_1X_2-1)^2 + (e^{-X_1} + e^{-X_2} - 1.0001)^2$

The test problem is a trough shaped function for which λ was varied from 10 to 1000 in steps of 10. Changing λ makes this function more badly scaled and thus more difficult to minimize. The limit on the number of calls was set to 1000.

Figure 2A shows that a ranking based upon one particular value of λ can be misleading. For $\lambda < 180$ the performance of OCOPTR and CGMIN are comparable, with OCOPTR somewhat superior. BFGS performs decidedly worse. However as λ increases the performance of CGMIN deteriorates rapidly. Furthermore, Figs. 2B and 2D demonstrate that for CGMIN and OCOPTR results are sensitive to starting points whilst Fig. 2C shows that the performance of BFGS is relatively insensitive to starting points.

c) Perturbed Quadratic Function
$F(X) = \Sigma iX_i^2 + \lambda(X_1^4 + 3X_2^4 + 4X_3^4 + 6X_4^4)$

This test function is a parameterized combination of a quadratic and a quartic function. When $\lambda = 0$, the function is a simple quadratic and is easy to minimize. However, as λ increases, the quartic becomes more and more significant. λ was varied from 0 to 1 in steps of .025.

Except for the case $\lambda = 0$, CGMIN proved superior to BFGS for this problem. Again OCOPTR performed the best and for $\lambda > 0$ was quite insensitive to the particular value of λ. Also, when the starting points were varied, the spread between the maximum and minimum number of calls was very small in all cases, indicating that the initial point is not extremely critical in this case.

6. CONCLUSIONS

Our primary aim has been to develop a testing method and software tools centered around parameterized test problems and graphical display of results, these being designed to aid the process of algorithm development. We have illustrated this method and shown the use of the tools in a simple case study. The results of this study provide interesting examples of how optimization routines behave when applied to parameterized test problems. The case study, of course, suffers from a number of drawbacks which must be remedied in a practical evaluation of optimization routines. For example:

a) Each routine studied employed a different convergence criterion. This introduces a lack of uniformity in the comparison.

b) A more detailed investigation should study each routine's performance initially, at intermediate stages, and in the final stage when it is near the minimum.

c) A broad set of test problems should be used, in particular problems with large or variable dimension.

d) For each test function more than one starting box should be used in order to sample a wider region of the topography.

ACKNOWLEDGEMENTS

Our thanks go to James Lyness for sharing his insights into testing with us and for a useful critique of this paper.

7. REFERENCES

[1] Lyness, J. N. and Kaganove, J. J., Comments on the nature of automatic quadrature routines, *ACM Trans. on Math. Software*, 2, 65-81 (1976).

[2] Hillstrom, K. E., A simulation test approach to the evaluation and comparison of unconstrained nonlinear optimization algorithms, Report No. ANL-76-20, Argonne National Laboratory, Argonne, Illinois.

[3] Fletcher, R., A FORTRAN subroutine for minimization by the method of conjugate gradients, Report No. R-7073, A.E.R.E., Harwell, England (1972).

[4] Davidon, W. C., Variable metric methods for minimization, AEC Res. & Dev. Report No. 5990, Argonne National Laboratory, Argonne Illinois (1959).

[5] Fletcher, R. and Powell, M.J.D., A rapidly convergent descent method for minimization, *Comput. J.*, 6, 163-168 (1963).

[6] Broyden, C. G., The convergence of a class of double-rank minimization algorithms, *J. Inst. of Math. & Applics.*, 6, 76-90 (1970).

[7] Hillstrom, K. E., Optimization Routines in AMDLIB, Tech. Memo ANL-AMD 297, Argonne National Laboratory, Argonne, Illinois (1976).

[8] Davidon, W. C., An optimally conditioned optimization algorithm without line searches, *Math. Prog.*, 9, 1-30 (1975).

Legend for Figures

a) Figures 1A, 2A, 3A: Average number of calls taken over set of starting points, plotted against λ. Superimposed plots are given for CGMIN(*), BFGS(@) and OCOPTR (#).

b) Remaining Figures: For each routine, Maximum (+), Average (0) and Minimum (X) number of calls for set of starting points, plotted against λ.

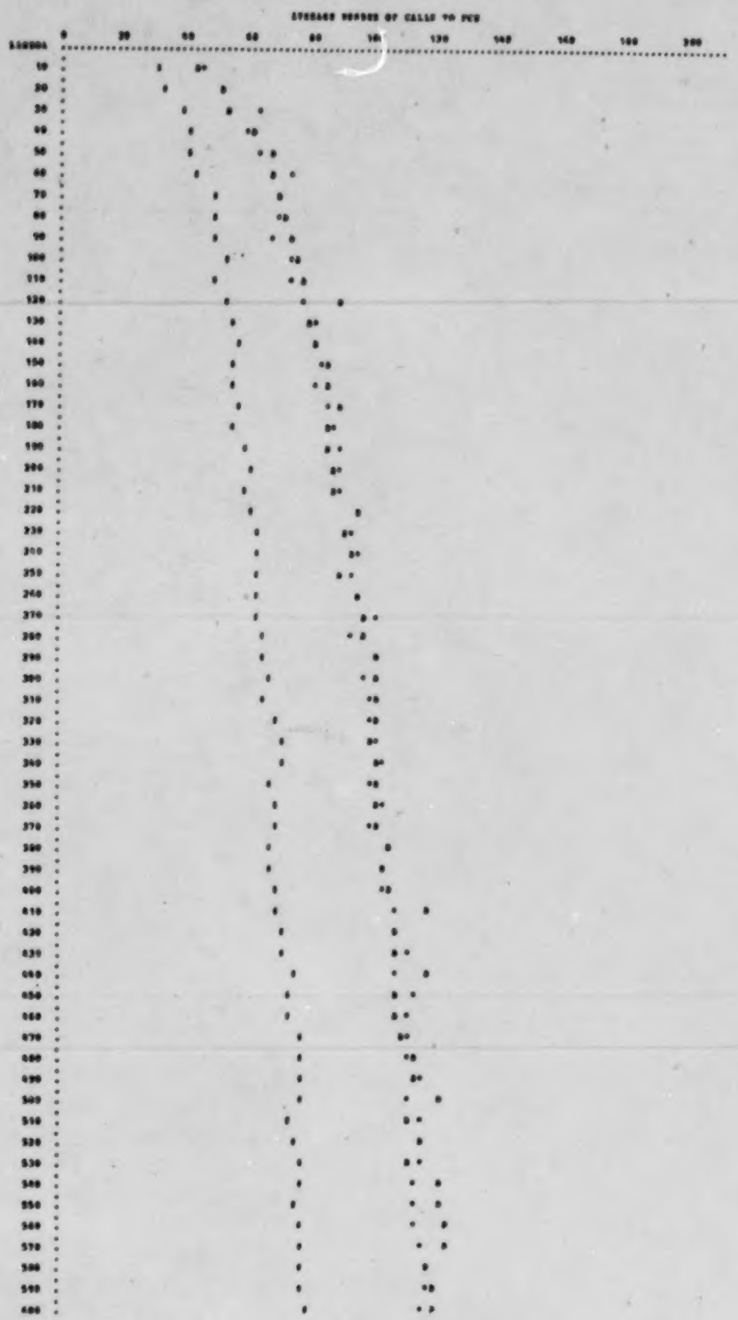


Figure 1A

Rosenbrock's Function - Average number of calls

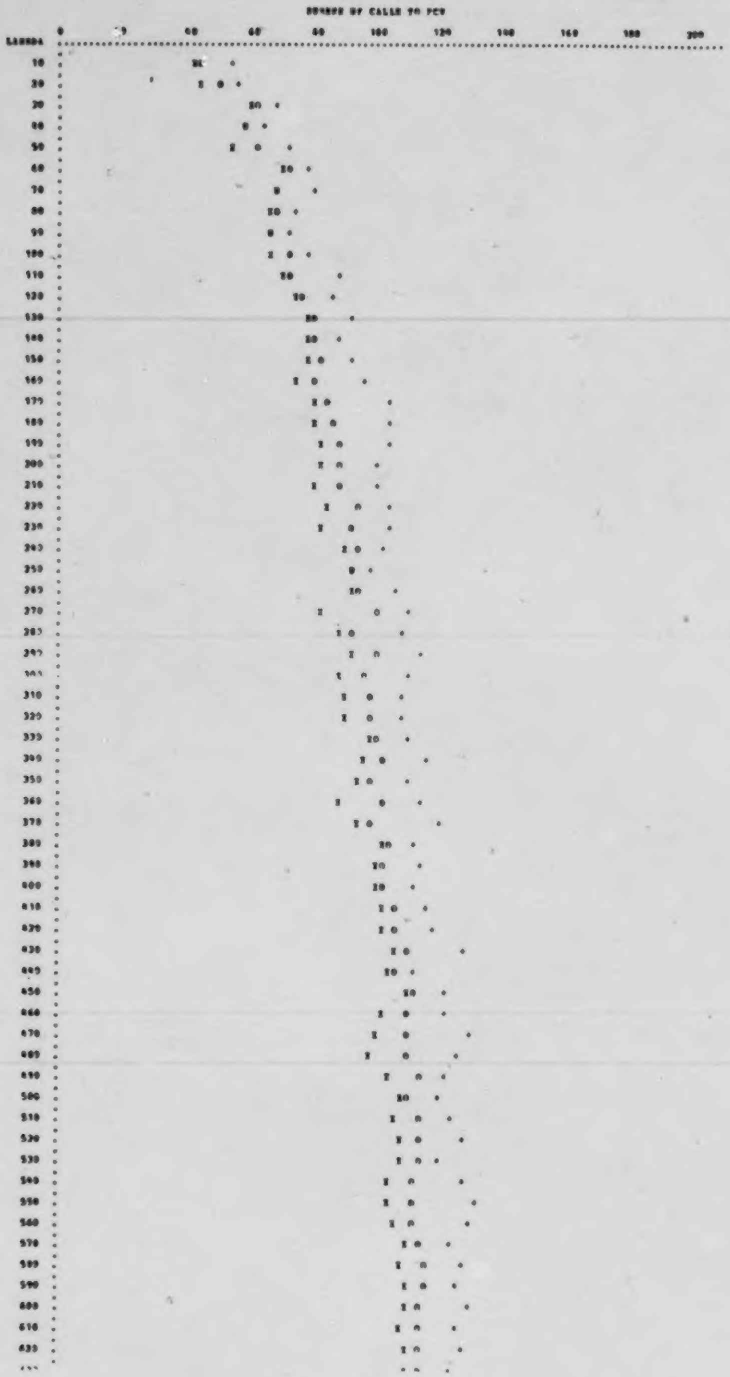


Figure 1B

Rosenbrock's Function - CGMIN

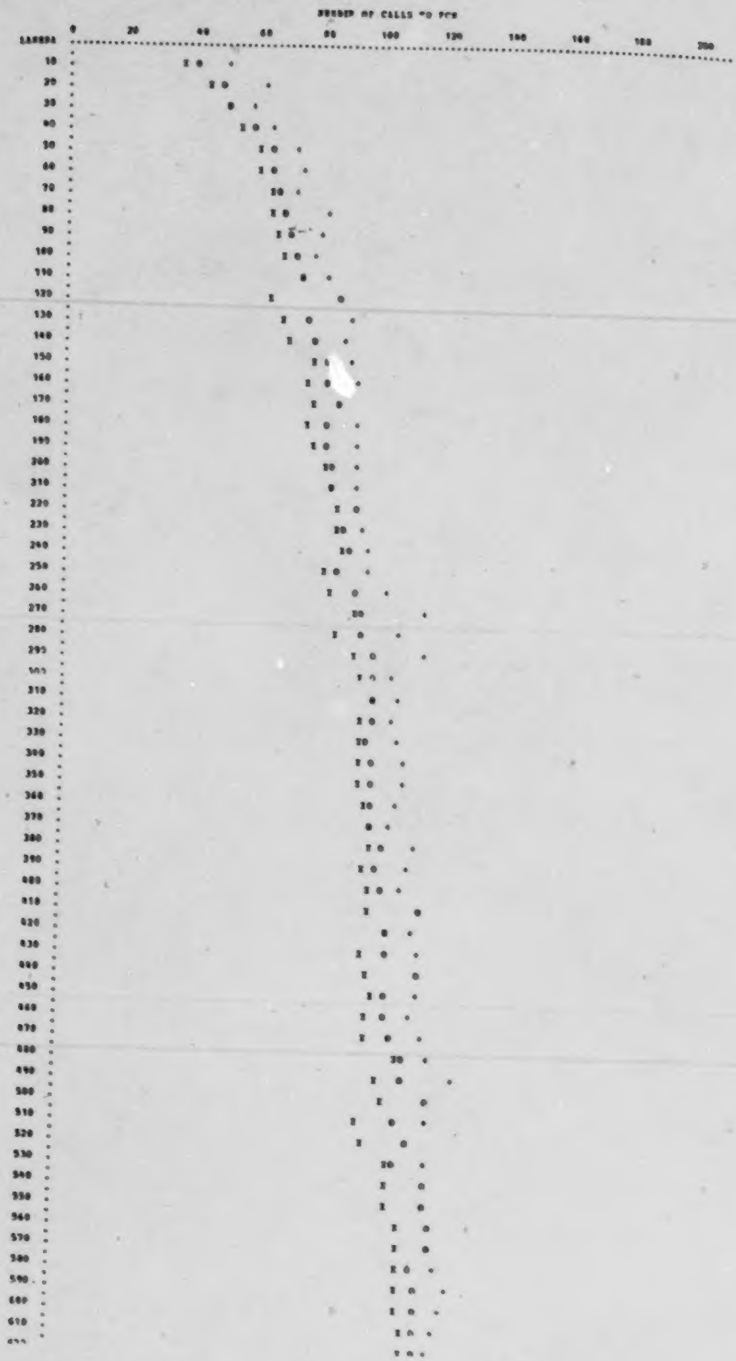


Figure 1C

Rosenbrock's Function - BFGS

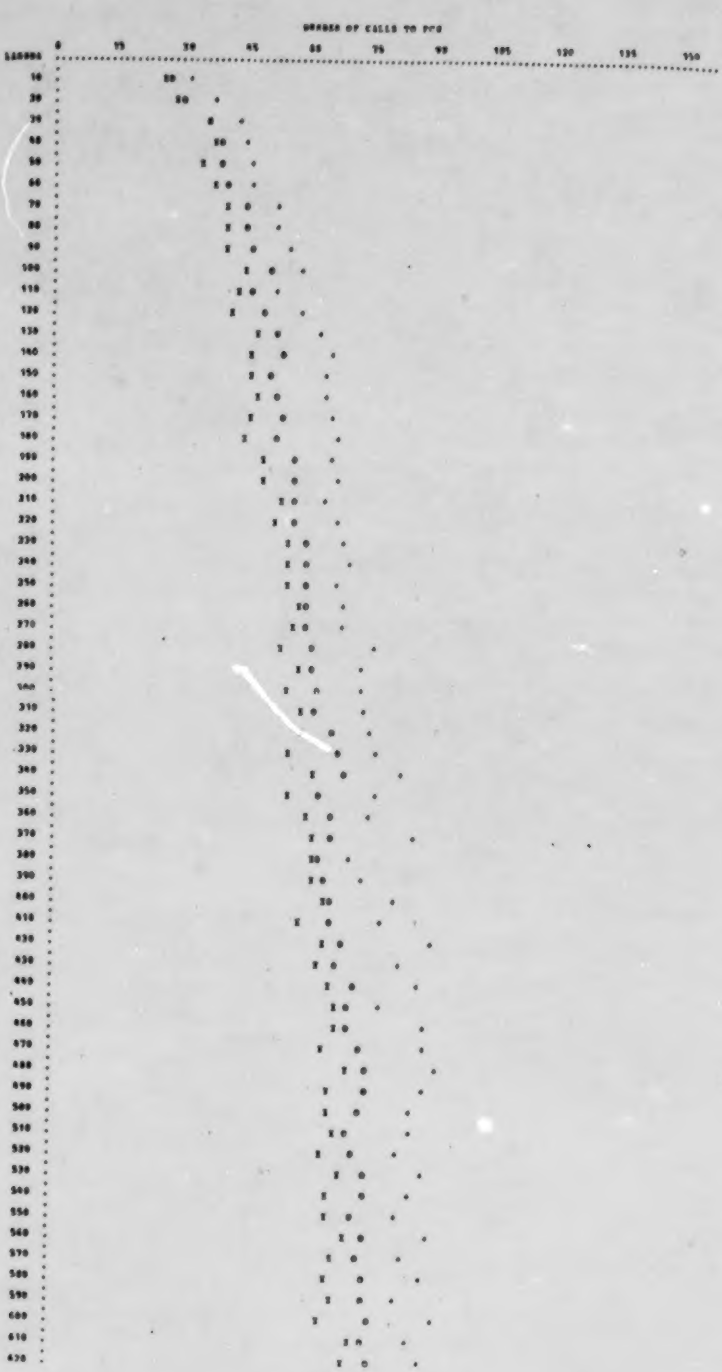


Figure 1D

Rosenbrock's Function - OCOPTR

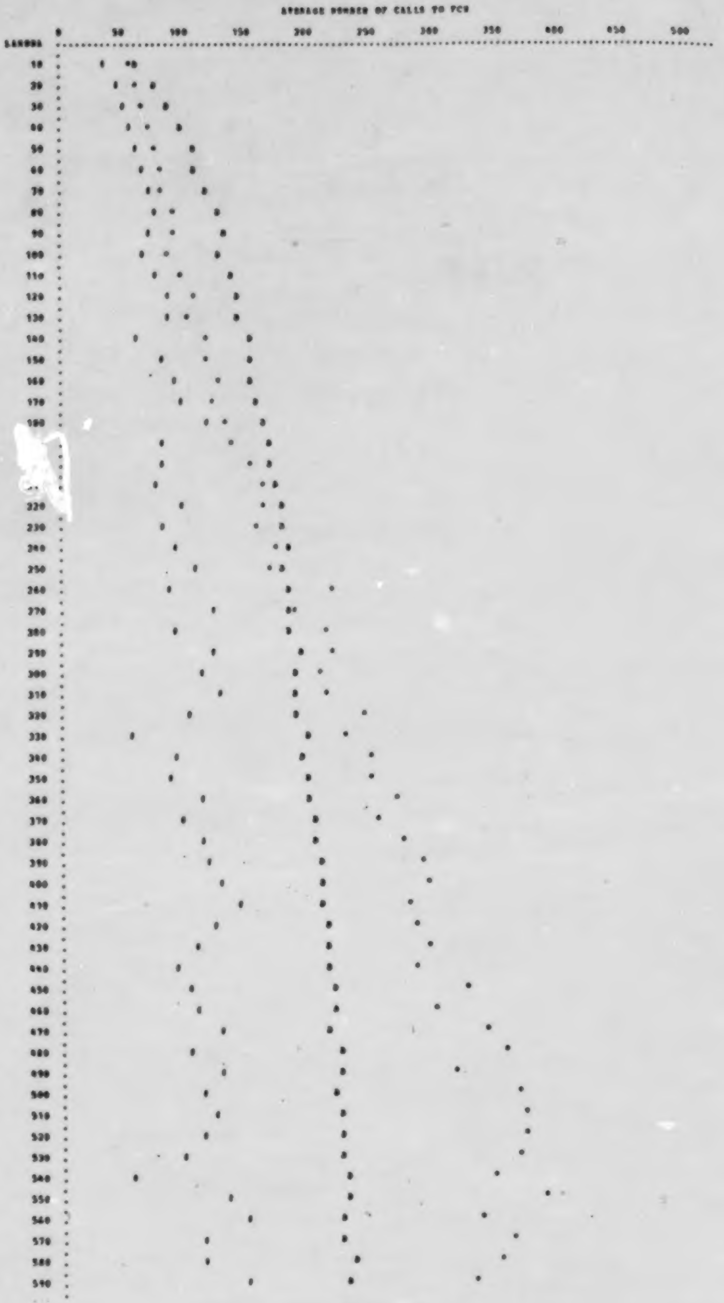


Figure 2A

Badly Scaled Function - Average Number of Calls

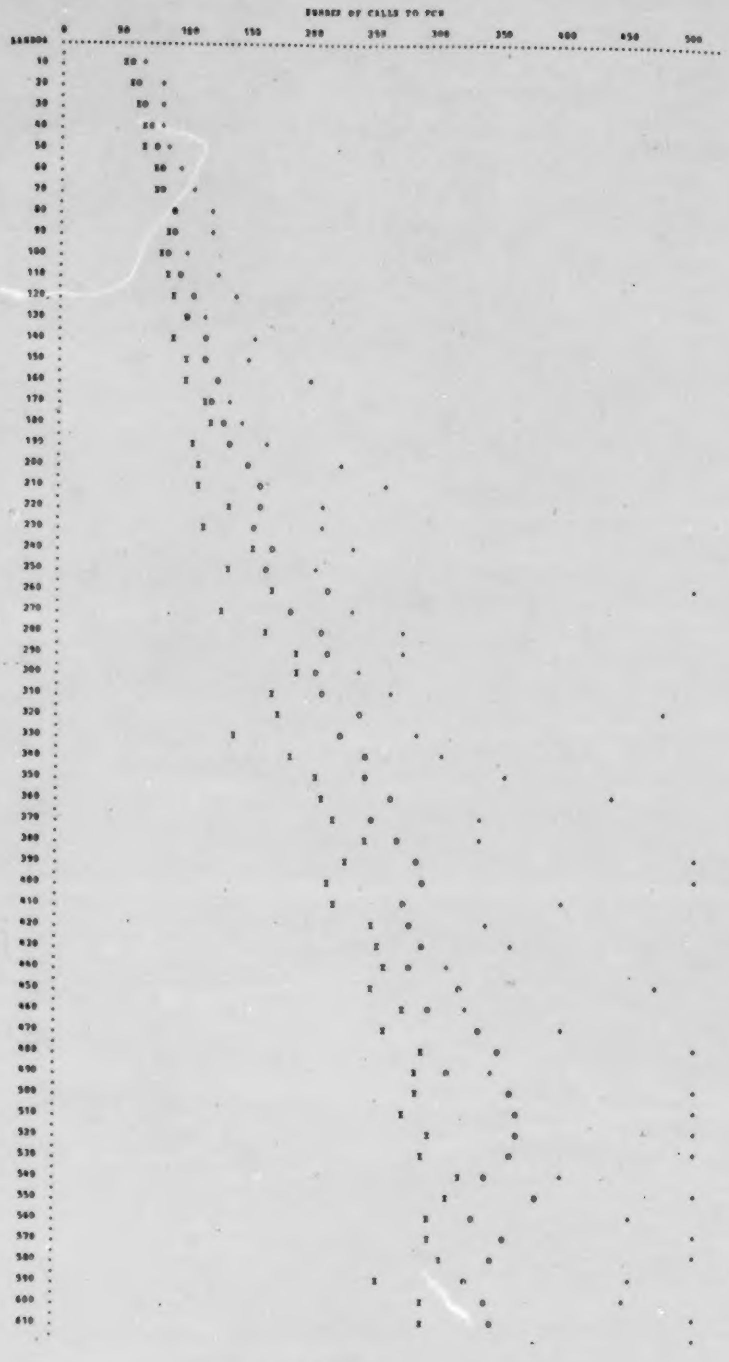


Figure 2B

Badly Scaled Function - CGMIN

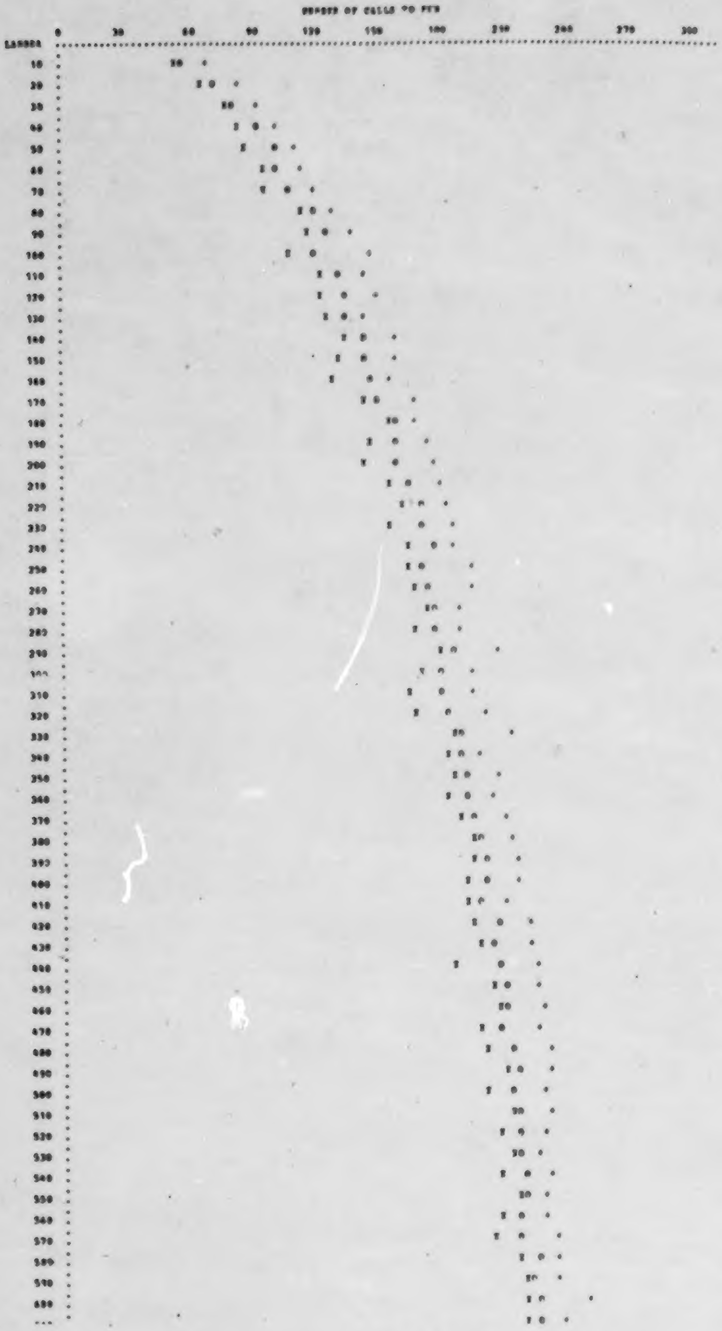


Figure 2C

Badly Scaled Function - BFGS

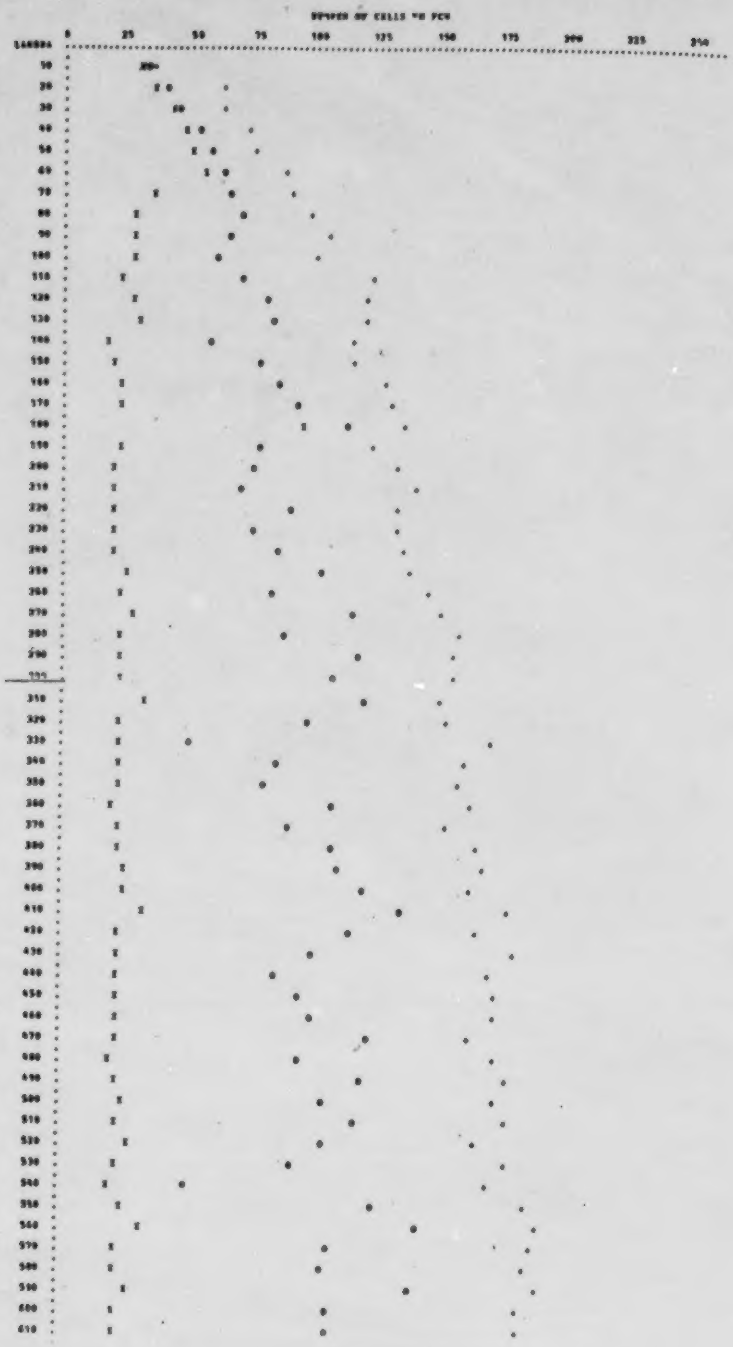


Figure 2D

Badly Scaled Function - OCOPTR

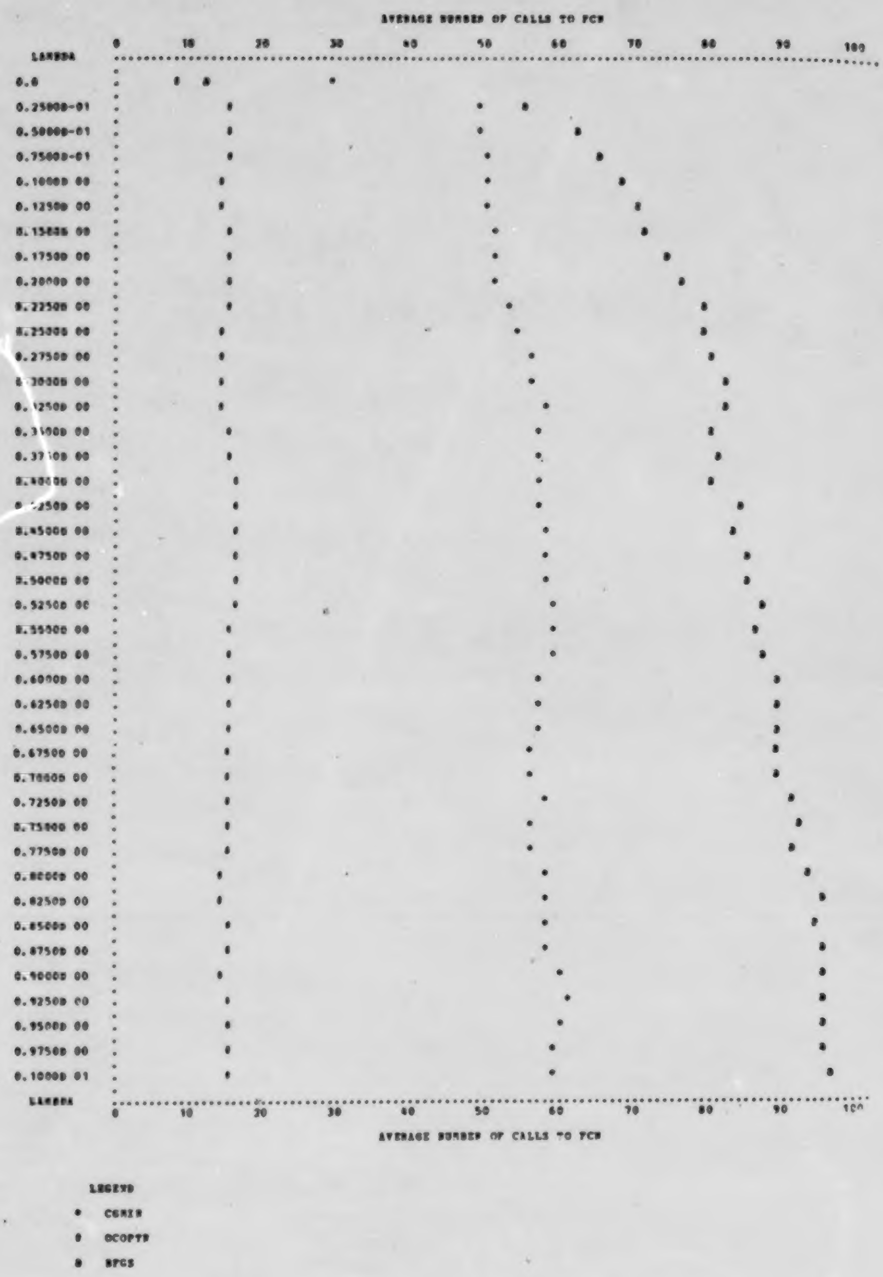


Figure 3A

Perturbed Quadratic - Average Number of Calls

142

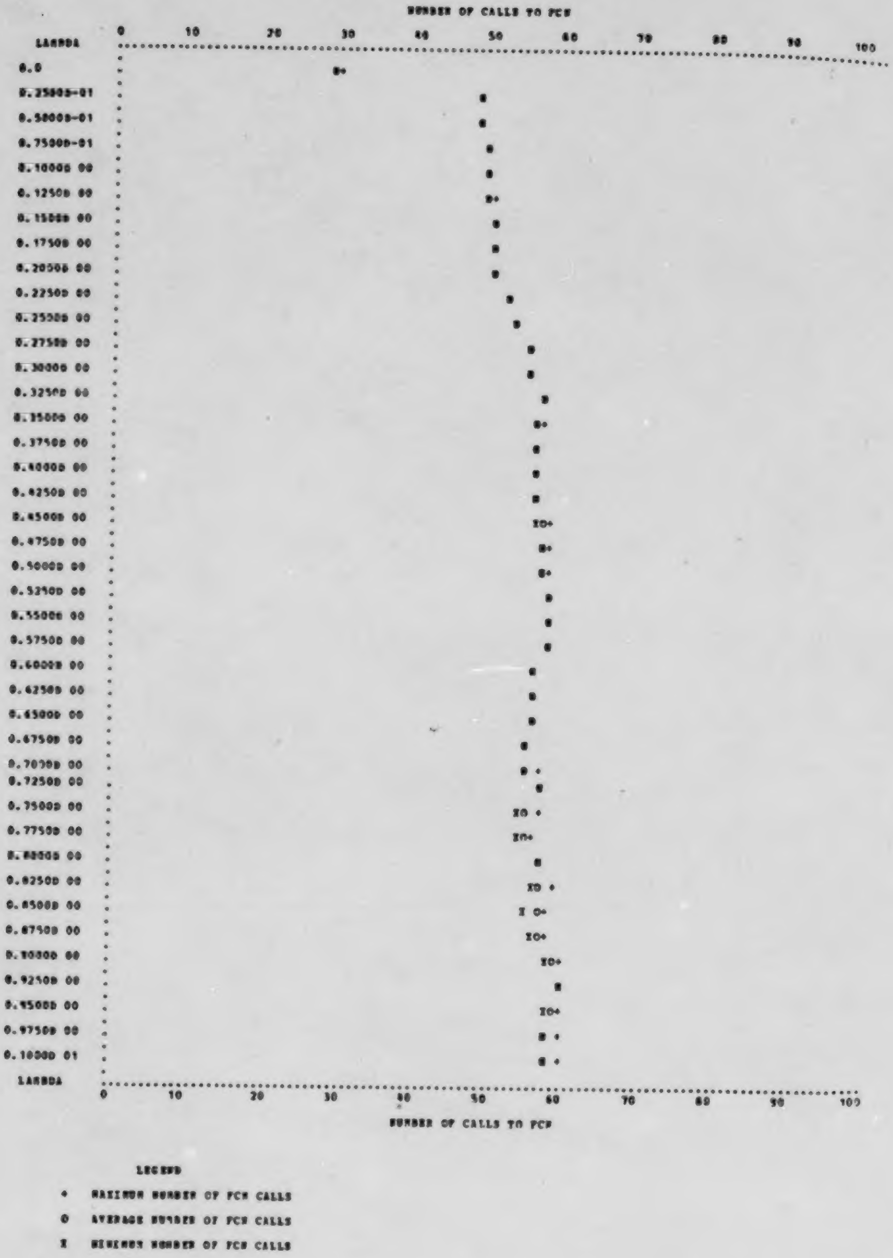


Figure 3B

Perturbed Quadratic - CGMIN

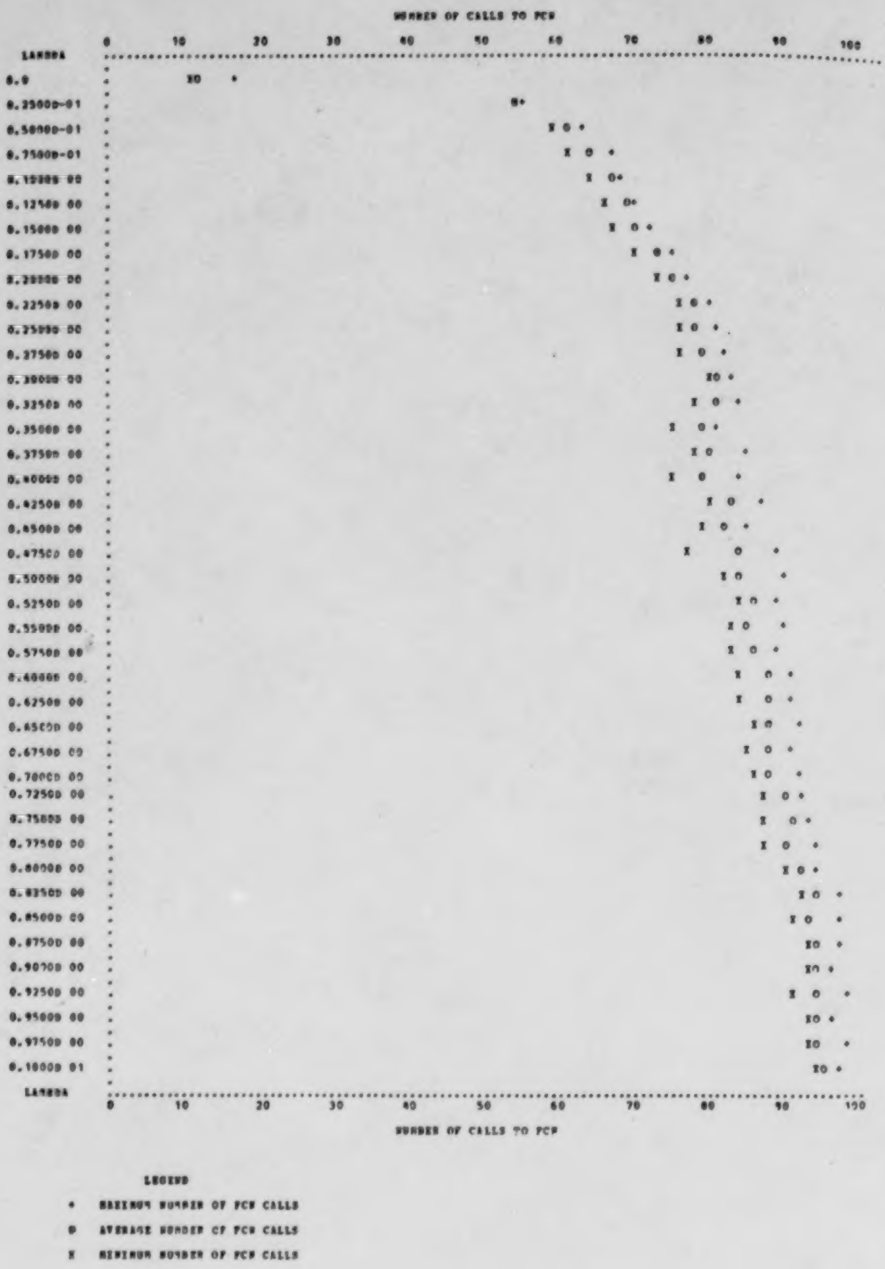


Figure 3C

Perturbed Quadratic - BFGS

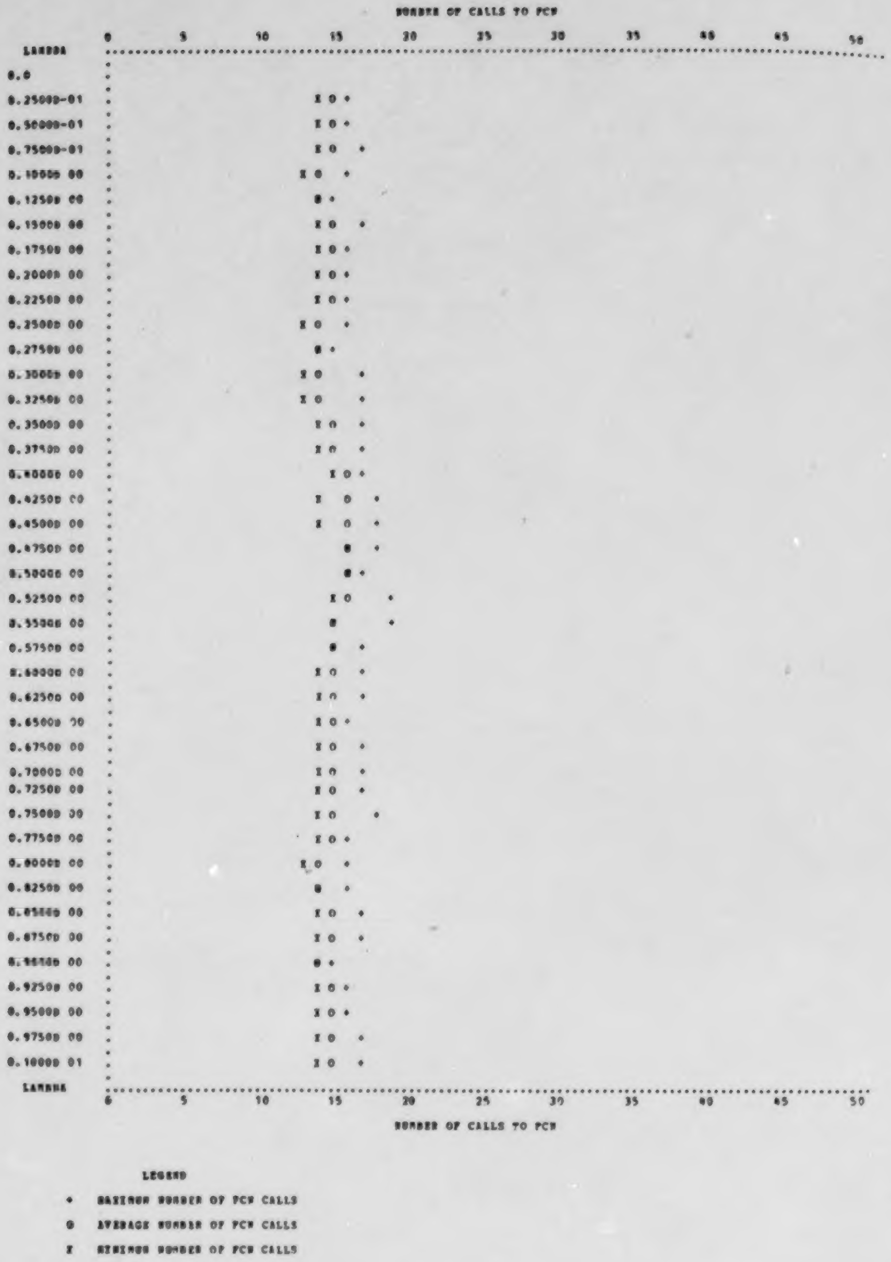


Figure 3D

Perturbed Quadratic - OCOPTR

"ISSUES IN THE EVALUATION OF MATHEMATICAL PROGRAMMING ALGORITHMS

PART 2: A PANEL DISCUSSION

CHAIRMAN

Richard H. F. Jackson
National Bureau of Standards
Boulder

PANELISTS

Ron S. Dembo, Yale University
Jerome L. Kreuser, World Bank
John M. Mulvey, Harvard University
Richard P. O'Neill, Louisiana State University
James J. Filliben, National Bureau of Standards
Harvey J. Greenberg, Federal Energy Administration

Provided herein is a summary of the comments made during this panel discussion. Although every attempt was made while summarizing to portray accurately the essence of each contribution, inaccuracies will persist. Responsibility for all such inaccuracies or misrepresentations lies with R. Jackson, who prepared this summary and functioned as chairman during the discussion. Names and addresses of all contributors to the discussion appear at the end of this text.

R. JACKSON: I would like to welcome you to the panel discussion on "Issues in the Evaluation of Mathematical Programming Software" sponsored by the Committee on Algorithms (formerly the Working Committee on Algorithms) of the Mathematical Programming Society. The Committee on Algorithms was created by the MP Society in 1974 with the charge to concern itself with knowledge, information, communications, recommendations, and other actions on mathematical programming algorithms and testing methodologies. The ambitious goals of the Committee were identified to be:

- ensuring that there is available to the MP community a suitable basis for comparing algorithms (e.g., a graded set of test problems);
- acting as a focal point for knowledge of computer programs available for general computations;
- recommending "best buys" where several codes are available for the same computation, and;
- encouraging developers to meet certain standards of portability, ease of use, and documentation (e.g., by producing "standards" or guidelines for the reporting of computational results).

This session, then, is being viewed by the Committee as another opportunity to create a dialogue between the committee and the mathematical programming community. We hope to focus the discussion today around the question of guidelines for the publication of the results of computational experiments, an item that I just mentioned as being one of our goals. With that then, I would like to throw the floor open for questions either from the audience of from the panelists.

R. DEMBO: I have a comment to make on the paper that Dick O'Neill presented in the morning session. In that paper, he conjectured that randomly generated problems tend to be more difficult to solve than real-world problems. It has been my experience that the reverse of that conjecture is true for Knapsack problems. I wondered if anyone else had a similar experience.

R. O'NEILL: I'll mention quickly that I can generate problems that will be solved in one major iteration by the Dantzig-Wolfe convex programming algorithm. So you can generate problems that are very easy to solve. I didn't want to make that conjecture as strong as it might have seemed, and I am sure that there are classes or sub-classes of problems that have been excluded, but some of the work we have done at LSU indicates that there is some truth to this conjecture. In addition, Darwin Klingman and Gordon Bradley also agree there is some truth to this conjecture. One point that can be made however, is that if you can develop a problem generator that generates difficult problems, you are in a good position to test codes that will eventually be run on real-world problems.

C. WITZGALL: In Jim Ho's lecture today on staircase problems, he mentioned that Martin Beale found staircase problems to be rather more difficult to solve than general, more distributed, linear programming problems. This could perhaps be a piece of evidence that works the other way, because staircase problems are highly non-random.

H. GREENBERG: But the proper comparison would be to have a generator generate staircase problems, so you would still be comparing staircase problems with staircase problems. It's just that one has random entries and the other has some real data. You would not compare a general LP with a staircase problem.

R. JACKSON: Is there a generally agreed upon definition of what a randomly generated problem is?

H. GREENBERG: I think it was at least alluded to in the presentations of O'Neill and Mulvey this morning, where they referred to specific structures with controllable parameters. But the situation is that you have a set of controllable parameters like sparsity and row counts and that the randomness appears in costs and right-hand-sides.

J. FILLIBEN: I would like to make a comment on the relationship between the typical math programming problem discussed earlier in the afternoon and a measurement process that one usually runs into in scientific experimentation. What you are really doing when you discuss what should and should not be controlled in a mathematical programming problem generator is specifying a measurement process. A well-defined measurement process has all of the components of that process identified; for example, the domain of variation. It should be noted that this is different from the simple injection of randomness into a procedure. For example, in the simplest situation, one lets everything vary with nothing under control, and from a statistical point of view, you are assuming least, and you end up with least. A better situation is what statisticians refer to as a "constrained randomization problem", where certain of the problem components are removed from the random domain and put into the deterministic domain. The remaining components that we can't control we then randomize. In essence, you have recognized that there are certain problem parameters that are of special interest, and in this case these are the ones that are to be controlled. What we can't control, we randomize. Two of the three talks that I heard this afternoon dealt with situations that could be described as constrained randomization. The other dealth with a completely random situation. But the point that I really want to make is that there is a relationship between testing by setting up problems in order to check out algorithms, with the corresponding problem in the scientific community of defining a measurement process by specifying exactly what conditions should be placed on the variables that we intend to control. It has to do with much more than just having a random number generator.

C. MYLANDER: It seems to me that in this discussion of using randomly generated or real-world test problems, we haven't made it as clear as we should just what the purposes of this testing are. It seems to me that we are testing codes and algorithms for two purposes. One is to have a code certified and safe for the user to use and the second is to compare algorithms for solution strategies so that developers of codes can decide what is the best solution strategy to build into a code. I would like to hear that kind of issue discussed a little bit more. From the point of view of a user, I'm willing to pay for computer inefficiencies so that I can use a small collection of codes rather than a larger collection of specialized codes. And then, on another topic, the one about efficiency of a code on a particular kind of test problem, I think my experience is that the way a problem is formulated causes much greater variance in computer efficiency than either the codes or algorithms themselves.

R. DEMBO: I think the two different objectives you mentioned, that of designing safe codes and designing robust codes, conflict to some extent. If you want to conduct experiments in order to have better equipment for designing codes, you should probably use the ideas presented in Larry Nazareth's talk this morning, because there you are testing specific properties of an algorithm. On the other hand, if you just take arbitrary problems and generate them, you have no idea whether you are testing to see if there is a very steep valley you are going into or what happens if you elongate the valley. So, if you want to design algorithms, I would say what you should do is pick problems by hand and perturb them. If you want to test robustness however, you would probably take as wide a class of problems as you can generate. But that means, of course, that if you are really interested in testing robustness, that's probably all the information you get.

H. GREENBERG: I think there are at least three purposes for doing any kind of experimentation along the lines we have been discussing. The first purpose has to do with program correctness: to see whether the program is correct. I really don't know how to do that, especially when you consider that the set of inputs that absolutely would guarantee correctness is larger than the total number of bit patterns in the input data. A second reason is to do benchmarking for the purpose of selecting an algorithm that you are going to buy to solve problems. I think the ϵ-perturbation method that Larry Nazareth proposed is a good approach to that problem. A third

147

reason for conducting such experiments is to discover the poorer aspects of a given algorithm so that that poor aspect can be fixed, thereby yielding a better algorithm. I should note that you want to be careful not to gear algorithm improvement towards a worn-out test problem, like Rosenbrock's function. I think too many algorithms have been designed to do well on Rosenbrock's function, especially Rosenbrock's algorithm, but it is not clear to me that using Rosenbrock's function will necessarily do anything for improving your ability to solve nonlinear programming problems in general. So I think that we have to take great care in designing a collection of problems that in some way contributes insights into algorithm improvement.

C. MYLANDER: I think we have to check the battery approach against sporadic selection of all sequences of test problems that are available to code developers.

H. GREENBERG: But in that case you only have a dozen hand selected problems with no indication of their properties or what happens when you start to vary things. I think the generation approach is more helpful in that avenues for research include getting a handle on a reasonably small set of parameters that is large enough to capture every aspect of problem structure that in some sense gives you reasonable representation of the real-world class being addressed.

D. SHIER: I wanted to follow along with the earlier topic of defining "randomly generated" problems I sometimes feel a little queasy about the definition of what is a randomly generated problem and what one actually gets out. I'd like to give you an example. Suppose I want to have five randomly chosen numbers that sum to one. One way to do this is to generate four of the numbers that sum to less than one and then subtract that sum from one to get the fifth number. Another way to do it is to generate five numbers randomly, and then divide them by the sum. However, that will not give you a uniform distribution over the space of interest. My point is that I think we have to look very carefully at what exactly constitutes a random problem. I also think that when you start to specify side conditions, especially the Kuhn-Tucker conditions for a given problem, there may be some problems there that I don't believe have been addressed by those who have produced test problem generators.

R. O'NEILL: I agree with you. In one of the experiments that I conducted, the only thing that changed in a problem from one run to another was that the cost row was generated by way of the Kuhn-Tucker conditions, as opposed to being randomly generated with values between 1 and 100. In one comparison, the Kuhn-Tucker generated problem took significantly more time, but in another case, the times were about the same. So I agree with you that we should try to determine the properties that these test problem generators have, but that's a difficult area.

J. FILLIBEN: I might mention this point: that there are a wide variety of statistical tests that one can use for checking various random number generators.

R. DEMBO: Let me add one thing to this. When John Mulvey generated the test problems for the paper we presented this morning, we discovered that another problem with generation comes into the type of sampling that you want to do as well. We wanted to do simple random sampling, so we had, in some sense, defined our population. We therefore picked a problem at random from that population. Now, we had each random variable varying uniformly according to certain distributions but I'm still not sure if what we did can properly be called simple random sampling. I wish I were sure of that.

R. O'NEILL: It's not clear whether test problem generators have any sort of inherent biases in them and I'm not sure whether you can ever determine that conclusively.

R. DEMBO: They do have a bias: feasibility.

H. GREENBERG: There are other kinds of biases too. There are generators that may be biased toward certain kinds of algorithms, so they would necessarily show that kind of algorithm in a favorable light. That's very little understood and I think one of the interesting avenues of research is directed toward understanding generator biases.

J. MULVEY: We have to be realistic about some of these things. Certainly it's very important to analyze the differences between real-world problems and generated problems but I think that as researchers if we're ever going to solve large numbers of problems, we'll have to use test problem generators. I think that for the future we can draw some parallels from some of the other scientific disciplines. For example, there is an American Society for Testing Materials that meets in this building, in fact, which is responsible for developing standards for testing materials. I believe that is something that will eventually happen to software: it will become an engineering function to test programs. But I think until such a society or institute is created, we're going to have to look at these portable generators for methods of comparing our algorithms. They're much easier to use than trying to shift large amounts of problems back and forth between researchers. I just don't think that's realistic at this time.

R. DEMBO: But that doesn't detract from testing the statistical properties of test problem generators.

J. MULVEY: No, in fact we should look more at the statistical properties of generators because I think that's the way the research is going to go.

D. SHIER: Let me add a comment to that. At one time we had a summer student who was doing some work on scheduling problems and had occasion to use one of the standard pseudo-random number generators. Turned out that whenever the number of items being scheduled was divisible by 2, very unpredictable results occurred. For example, regardless of the number of items being scheduled, in some cases he had hundreds of classes, it all could be scheduled within two hours. It turned out that

2 was the seed of the random number generator. So that sometimes it comes and hits you on the head that it really is necessary to dig down and look at the random number generators that we're using.

J. FILLIBEN: This brings up the point that a given random number generator is only "random enough" by comparison to the purpose for which it is to be used; and it sound like, for mathematical programming test problem generation, there are some strong dependencies on very subtle properties of random number generators that are being used. Furthermore, these properties may in fact depend on the algorithm on which the problem is to be run. So that a given random number generator may be alright for a certain type of algorithm but not alright for another type of algorithm in the worst possible case. So getting back to my original point, I'd like to emphasize that whether a given set is random enough really depends on for what purpose that set is to be used. This in turn implies a need for a thorough understanding of the various kinds of mathematical programming algorithms and the properties of each algorithm as they relate to test problem generation. I don't think that such understanding exists yet.

R. JACKSON: This appears to indicate, then, that in addition to investigating algorithm performance on particular classes of problems, we should also investigate the correlation between a particular algorithm and the particular random number generator used in the test problem generator that produced the problems on which an algorithm will be evaluated. And if this is true, then we have identified another avenue for research in evaluation methodology.

R. DEMBO: I would like to suggest that we move on to an important topic with immediate consequences: the issue of guidelines for publications. I think it's very important that we tighten up the criteria used for selecting articles for publication when they contain computational results. I would like to put it out as an open question for the audience to inform us if they have any suggestions as to what guidelines could be implemented immediately. Certain guidelines are, to my mind, obvious. For example, specifying a compiler when you run a problem; specifying convergence criteria when dealing with nonlinear programs, but I could go on and on.

R. JACKSON: This general issue of guidelines is a very difficult one and I agree with Harvey that trying to discuss the general question of publication guidelines might be difficult. However, certainly there are aspects of that question that we could get into.

J. MULVEY: I think we could go right to the question of test problems. For example, should we require researchers to provide test problems to other researchers when they publish results? I think that is a question that we can pose for reproducibility.

R. JACKSON: Before we move into the discussion of the collection of test problems, I would like to provide a little background. In the first place, we are assuming that in the absence of any other well accepted method of comparing one algorithm against another, the test battery method is a valid one. Furthermore, given the fact that the test battery method is an acceptable method, then there ought to be produced a well accepted set or perhaps even a graded set of test problems that are to be used by all researchers. So the question becomes then, how can such a set of test problems be created? This has been looked into by a number of groups over the years; SHARE did that, Wolfe did that, along with many others. The question confronting the committee now is what kinds of ways are available to produce such a graded set of test problems? One suggestion is to require that anyone publishing a paper containing a computational comparison of codes should make those codes available to all other researchers and perhaps even a central facility should be created for storing them.

H. GREENBERG: I think the question that has now been brought to the floor is not guidelines for running tests, but guidelines for publication of tests. We are narrowing the subject to just the issue of publication criteria where the contribution hinges on the claim of a better algorithm. One of the proposals then is that no one should be allowed to publish results claiming better performance of an algorithm on a set of problems that are not in the public domain. More specifically, whatever problems have been used by researchers must be made available either for the sake of reproducability by referees or for other researchers to try out their methods on the same set of problems. In short then, it's the publication issue that's been put on the table, not the general issue of guidelines.

R. O'NEILL: Can I confound the problem? How about providing the code itself?

R. DEMBO: I suggest that we attack the easier things first before we get on to the trickier problems like exactly which guidelines you would suggest for publication. There are certain standards that could be implemented immediately. For example, reproducibility: a referee should be convinced that an experiment could be reproduced. That's an example of what I think is an important criterion.

R. O'NEILL: I could add that I think all of the important controllable parameters should be included like machine, compiler, operating system. It is not necessary to pile up the paper, but this could be included in an appendix.

H. GREENBERG: I'd like to point out that in the early days of the development of the SIMPLEX method, if very severe restrictions had been placed on the publication of results, it may never have gotten published.

J. GILSINN: In these days of page limits per article, trying to include a listing of a computer code is probably not a realizable idea, but perhpas it is possible to require that enough background information be included in such an article that would then allow reproducibility if one established contact with an author. Another choice would be to provide some central repositories where the codes could be obtained. But I think there is a conflict here that must be resolved.

H. GREENBERG: I think you have a line drawing problem because some codes have thousands of lines and it is not possible to include much information about serious large scale systems.

J. GILSINN: Yes, but I also want something more than just so and so's algorithm. I want information about how that algorithm was implemented. I want to know information about what research techniques were used.

A. WILLIAMS: Both problems have been handled simply by having the author include a statement that a listing of his code is available. A number of the journals are encouraging authors to do this kind of thing.

H. GREENBERG: What do you do then about proprietary systems where it's informative to get into a journal some analysis but where the owners of that system are not going to release their proprietary information about the code in the form that you're discussing. Most certainly they're not going to make available source listings of such systems as MPSX. Can we eliminate that from the journals?

A. WILLIAMS: In my opinion, seriously, we wouldn't eliminate it, but we would charge them an advertising fee.

H. GREENBERG: I think it's not a binary situation. I think that there is much useful information that can be contained in an article without requiring that a listing be associated with it.

R. DEMBO: When I originally mentioned this, I didn't have in mind a requirement that authors submit listings. My concern was with requiring referees somehow to convince themselves that given a listing, they could reproduce the results. They had to convince themselves of the integrity of the authors.

H. GREENBERG: I don't think that integrity is the issue. I think the transmittal of information is the issue.

E. HELLERMAN: In this connection, what worries me is that when comparisons are made, they're made against your code and someone else's code. Very frequently, it's someone else's code that is at stake and in this case we don't know what kind of implementation that person has of that other person's code. This is worrisome.

R. JACKSON: I think we should keep in mind the purpose for requiring additional information about algorithm or code. We're seeing lately that it is no longer sufficient to run a few problems on two codes and report overall CPU times which are then used to support claims of superiority. It is becoming more and more important to understand what is going on in the interior of the codes that are being compared. Knowledge of pivot strategies, for example, is important. And if you deal with proprietary codes then there is no satisfactory way to get information about what kinds of techniques are used in the algorithm of which that code is an implementation. In this case then, we're left in a situation where that code simply cannot be compared.

J. MULVEY: If you really want to be strictly scientific, I think you have to have a code available because we're testing codes, not algorithms. I think we've thrown about the words algorithm, code and software a little haphazardly and that in our experiments we deal with codes and make inferences about algorithms. So that, if we're going to be scientists, those codes should be available. However, I don't think that is a feasible suggestion right now. There will be people who are unwilling to make their codes available and that would just shut off what fearful little results we have now; at least reduce it quite a bit. I think we can work slowly toward requiring more and better information to be provided either through an appendix, or by way of the authors themselves. Our problem then, is to determine how to increase the informational flow.

H. GREENBERG: I think it has to be acknowledged though, that there is a trade-off. The goal is to increase the transmittal of useful information but if you impose restrictions saying "You will report this", that doesn't necessarily result in an increase of information because the reply can be "No, I won't."

R. JACKSON: The situation then becomes what Al suggested where authors should be charged for advertising. Papers would be appearing that are essentially a claim of superiority with no supporting information allowing a replication of the experiment or even the checking of the types of strategies and techniques used within a code. I'm not saying that is not transmitting more information. I'm simply saying as a code comparison or a claim of superiority, there is much room for improvement.

J. MULVEY: I think there's an analogy to be drawn in consumer unions in the sense that they go out and test and break, and produce good results on that empirical evidence. In our case we could have a group that goes to a particular installation with a battery of test problems and says "Here, solve it on your machine, on your algorithm, etc." The big question is: who's going to do that work? That's not clear.

E. HELLERMAN: I think a bigger question is: who's going to pay for it?

C. MYLANDER: The suggestion that ought to be put forward is that requirements for publication ought to be that the test problems and the hour that the test was run are fully specified. I don't think that it's necessary to specify the codes also. In this way, if I have a code to solve a class of problems, I could run on exactly the same test problem if I wanted to go out and rent time on that same computer and run under the same operating system so that a valid comparison could be made.

J. MULVEY: I think he used an adjective that forces you into the situation of sending out codes and that was that the experiment must be fully described. Fully described means that you have to have a code. You can summarize it, certainly, but you will be losing information.

UNKNOWN: It seems to me that we have two different items here; algorithms and codes, and that they're completely different. If someone wants to say "I have a code and it's a good one but I don't want to tell you what's in it" that's fine, so long as he presents the problems that he ran and presents his tests along with the environment in which he ran it. If it's an algorithm he's pushing, it's a different story. He must describe the algorithm behind the code in detail to make it valid, and he must also provide a listing.

H. GREENBERG: I think a good example is the Hellerman-Rarick invert routine, where the invert code was not made available for free, but it was nevertheless a valuable contribution to the literature to have it published.

A. WILLIAMS: Why, though, would something like that be published as a scientific paper rather than as an advertisement or flyer?

H. GREENBERG: I don't think the name "Management Science Systems" meant anything to someone reading the article so I don't think it was an advertisement. I think rather it was a major contribution to understanding how reinversion should be done.

A. WILLIAMS: I'm not familiar with the details. Did they speak broadly about what was done, not specifically?

H. GREENBERG: No, they gave specific algorithms but the implementation of the algorithm and the code was not specified and that makes a big difference when you try to duplicate.

A. WILLIAMS: I guess I don't understand. When there's some kind of a mathematical procedure, unless you knew the real trick when coding it, you mean you could not code it?

H. GREENBERG: No, you could code it. I coded it. But you almost certainly don't have as good an implementation as the one Dennis Rarick had. So that if you tried to compare MPSX's invert routine with your homegrown code, you won't necessarily be comparing like items unless you're as skilled a programmer as Dennis Rarick was. Very few people are. Nevertheless, the whole algorithm is there and there's enough information for you to code it. On the other hand, it's not just an algorithm. It offered a whole new concept of looking at reinversion. The idea of a spike was introduced in that paper which has become a classical paper in the literature of mathematical programming systems. Much subsequent research has taken place because of it.

J. MULVEY: Why does the company allow you to publish an algorithm which gives the essence of an idea and not publish the software. It seems to me that both the idea and the implementation are proprietary.

H. GREENBERG: It's for precisely that reason. Because the full power of the method rests upon such clever coding that they were unafraid of the competition and allowed it to be published.

E. HELLERMAN: Actually, P3 was described in terms of ALGOL. There were ALGOL-like statements describing every facet there. Also, I've gotten reports from a number of universities stating that they implemented the algorithm strictly on the basis of what we had written in that report, P3. It was clear cut that they could write their own code and have it working almost immediately.

M. GUTTERMAN: Actually the code itself is dependent not only on the machine, but on the data structure with which you happen to choose to represent your LP matrix and LP inverts.

UNKNOWN: We're actually introducing a third problem here because the code structure is another dimension completely.

H. GREENBERG: Absolutely. You run into that all the time where a major revision in the data structure has a greater impact on the resulting performance of a code than a major change in the algorithm tactics.

M. GUTTERMAN: I can testify to that one personally since I've been involved in looking at the results of three separate implementations of the Kalan Matrix Packing Ideas on the same computer.

R. JACKSON: Perhaps we're getting to the point where we ought to broaden our definition of what exactly an algorithm is.

R. DEMBO: No, I think my original comments referred not to whether an algorithm was published in coded form or not. I agree with the comment, by the way, that if you publish the code it would mean less than publishing a mathematical description. But let's take the paper P3 as an example. If claims were made in that paper that the new P3 invert procedure was much better than what had been done previously, the question then becomes whether enough information was given in that paper to allow those claims to be tested by someone else willing to code the invert procedure. It must be understood that these comments are made with the understanding that this new researcher might be a worse programmer than Dennis Rarick was.

J. GILSINN: I'd like to go back to a previous topic of conversation. Are you thinking in this set of guidelines of not allowing a paper to be published if it presents computational results about a code that is a proprietary one? Because it seems to me that users of codes are most interested in knowing the performance of a particular code against one of the better known codes and very often these better known codes are the proprietary ones. Whose going to restrict that kind of a situation?

H. GREENBERG: I think the correct answer is that we don't know.

R. JACKSON: The issue is getting a little more complicated, and I'm not sure we're defining our terms very well. Our original topic of conversation was the development of a set of guidelines for the publication of computational results where the ultimate aim is to see better comparisons made. We went off into the tangent of discussing whether the code itself should be published as a result of the comment that if you don't see the code you can't do a proper comparison. Another issue has been raised, however, that in the absence of complete listings of the code for whatever reason, proprietariness or unwillingness of journal editors to include listings, is there anyway that a fair computational comparison can be made about codes? I would like to see that topic discussed moreso than the current topic of whether codes should be published or not. It is clear that proprietary codes are not going to be published.

W. ORCHARD-HAYES: But related to that question is the argument that there is no such thing as reproducable complicated coding. It depends on too many factors including the style of the author, the compiler you're using and whatnot.

R. DEMBO: I didn't mean reproducing code, I meant reproducing experiments.

W. ORCHARD-HAYES: No, I mean reproducing results.

R. DEMBO: Well, I think you're right. There are a lot of different factors that enter into it, but you should be able to attack the same problems, using the same tolerance criteria as the authors did and produce the same results.

W. ORCHARD-HAYES: I guess the way I'm reading it, it's a question of relevance. What difference does it make?

R. JACKSON: The answer to that revolves around the question of replication as a necessary part of scientific endeavor.

R. DEMBO: The point is that you're making inferences.

H. GREENBERG: I think the payoff is a bit better than that now. I think it's hard to understand when there are no controls. If one person runs an experiment with one problem and someone else runs another experiment with another problem, there is no way of knowing whether you've made any progress. I think the value of reproducibility, to the extent that it's feasible, is that it does give you a measure of progress.

J. MULVEY: I think there is some value in saying why one code is better than another in that when someone else tries to duplicate the experiment they can at least have some systematic way of trying to get the same kind of results.

H. GREENBERG: I think it's true for the opposite reason, namely to inject the objectivity that make it less dependent on the judges. I think that's the point of reproducibility.

E. HELLERMAN: I think there's an awful lot of pure artistry here, artistry in implementation. Now, I would stack anything that Bill writes against anything that anybody else writes, and I know it's going to be better because Bill's an artist at this kind of thing. It's like looking at a painting of the same landscape by two different artists. They're going to look a little different no matter how hard they try to be the same. I don't know if you can develop a criteria for pitting one against another to determine which one is better.

R. JACKSON: I think I would like to disagree with you on that. I agree with you that there's an incredible amount of artistry in producing some of these codes, but artistry, I think, once it is around long enough and used over and over again is no longer artistry. It becomes documented fact. Let's take list structures as an example. At some point in time years ago, how data were stored, organized and retrieved was almost a black art. But now techniques for organizing data and retrieving it appear in textbooks. And this is getting back to what I mentioned awhile ago, that it might be necessary to expand our definition of an algorithm to include such items as specific coding tricks.

W. ORCHARD-HAYES: Why not just say "list/structure" then? Why isn't that sufficient?

R. JACKSON: My point here is that there are probably other aspects of what is now artistry that should be included in this expanded definition of an algorithm.

H. GREENBERG: I agree that what was art ten years ago may be partially science now.

W. ORCHARD-HAYES: I'd like to make one more point about comparing codes and that point has to do with portability. I just can't believe that it's possible to carry one code from one machine to another machine and compare it. Codes just aren't that portable anymore.

H. GREENBERG: Right. We have identified machine characteristics as a set of variables to be reckoned with in the design of an experiment. I think through scientific investigation we can get a much better understanding of the various computer and algorithmic effects on the results of an evaluation. That, of course, is the point of the investigation: to understand the effects of the variables. It's my feeling that there will be some interesting results from the work done recently by Dick O'Neill. I believe there will be some very strong and counter-intuitive results from that work.

M. GUTTERMAN: I personally don't believe in printed publication of codes. I think that code availability by publication, the creation of a collection and dissemination center for machine readable code, is a valuable contribution in many cases. But despite the artistry of Bill Orchard-Hayes's coding, I've never gotten much benefit from having a listing of it in front of me.

R. O'NEILL: You know, if you chose the test problems correctly, you could actually design an experiment to test one person's artistry in coding against another's. You could in fact make inferences about artistry.

UNKNOWN: This sort of thing is being done and results have been reported in a book called The Psychology of Computer Programming. The experiments were designed to test whether a code produced by different people varied according to the stated goals. It made a tremendous difference whether the stated goals were to produce code as quickly as possible or whether the code was meant to be as efficient as possible. In any event, this kind of experiment is being done and the results are very interesting.

E. HELLERMAN: I'd like to point out something here about characteristics of computer programming. I actually recall an instance where one of the oil companies was approached by Bonner and Moore and informed that their problems could be running in one third the time that they're currently taking. Bonner came to our installation to run it and spent a lot of time fiddling around with the formulation of the problem until he finally met his objective of solving the problem in one third the time that it was currently taking for that oil company. But the reason that he was successful is that he knew enough about the characteristics of his code and what is required in the way of the characteristics of the formulation, in order to take maximum advantage of his code characteristics. So these are a few other things one runs up against when trying to measure code performance.

M. GUTTERMAN: Another question to be answered however, is what would have happened if they had taken the new formulation of the problem and put it on LP90 - would it have run faster in that case?

E. HELLERMAN: Another important point, though, is that Bonner knew enough about the oil company's problem to say that if you can live with an error larger than was being allowed by LP90, you can get the kind of reduction in time that he was talking about. And he indeed came up with a solution that had an error at the maximum rate, a rate that would never have been tolerated by LP90. The question then becomes what can you live with? How large an error are you willing to tolerate?

R. JACKSON: That question of what you can live with is a difficult one and makes me think of the comment Chuck Mylander made earlier about wanting to keep only a small selection of codes. There are people that I've done work for who want only one nonlinear programming code. One man doesn't care whether there's a few seconds difference between the code that he's got and the other codes that he might be able to use. He wants only to learn how to operate and become familiar with one code rather than have to deal with a large number of them.

C. MYLANDER: As a referee, I still get alot of papers in which an algorithm is proposed but the paper doesn't have any other merit than the proposed algorithm and it hasn't even been coded.

H. GREENBERG: Why don't you reject it?

C. MYLANDER: I do. It's just that I think it would be a good idea to propose a standard for people who publish algorithms and computational results making it mandatory that they have run the code, specified some test problems, and listed the environment in which it ran.

H. GREENBERG: The evolution I see, for example in nonlinear programming, is that fifteen or twenty years ago we were pretty hungry for algorithms making it possible for an algorithm to be proposed with no more justification other than that it was a clever new idea. As time went on, however, it became necessary to provide empirical evidence, or theoretical evidence like rate analysis, about the efficiency of a clever new idea. At that time, say in the sixties, it was acceptable to do an experiment consisting completely of a randomly generated problem. In the seventies, however, we're discovering that is no longer acceptable and there appears to be a move toward much more sophisticated design of experiments to test whether an assertion of a superior algorithm is true.

H. CROWDER: What's the possibility of sending programs and listings to referees that can be used in refereeing the paper. For example, I once was asked to referee a paper for TOMS. The editor, Milt Gutterman, sent me a deck, and a listing, and the third example I tried on the code failed. I simply packed it all up and sent it back to the editor and it's back in the hands of the author now.

R. DEMBO: When I originally brought the topic up, I was thinking about the kind of article that's published in which the authors say "Here is a new algorithm that took two seconds to solve Rosenbrock's function and therefore this is a really neat new algorithm. That kind of article is still being published, for example, in Mathematical Programming. It shouldn't be. We are now at the point where we need much more information than that it took two seconds to solve Rosenbrock's Function.

UNKNOWN: One thing I haven't heard mentioned today is the situation where an algorithm is proposed to solve a previously unsolved problem or for some other reason there are no other algorithms to compare against it. My question is should the proposer of the algorithms be required to program it? There may be a number of mathematicians who simply are unwilling to do that.

H. GREENBERG: As I understand the spirit of the guidelines, they're based on the assumption that they will be applied only for papers whose primary contribution is based on a claim of superiority or some other way related to competition with other algorithms for solving problems. Only in that circumstance are the guidelines applied and the authors are asked to satisfy certain experimental design criteria. Not much thought has been given to the situation you're describing.

R. O'NEILL: Before the session ends, I would like to ask the audience a few questions about some of the topics we've discussed today. I'd like to know, for example, how many people here think that the computer hardware should be mentioned or specifically included in a paper to be published about computational results? What about the operating system? The compiler? The specific test problems used? Should the test problems be made available? I think we have a consensus.

H. GREENBERG: One more question. How many care whether the publications carry that kind of information?

R. DEMBO: It appears, gentlemen, that we have won.

R. O'NEILL: One more question - how about the code? How many people think the code should be made available? No consensus. What about available, but not published? No consensus. What about available to the referees only, for evaluation? A consensus.

M. GUTTERMAN: That's a suitable compromise.

R. O'NEILL: What about proprietary codes? Should they be made available to the referees only for evaluation? No consensus on that topic.

R. DEMBO: We've asked a lot of questions here and got some very good input from the audience. The next question we have is what exactly should we do with it? How do we go about getting these ideas implemented?

J. FILLIBEN: Isn't that a problem for the editorial boards?

H. GREENBERG: Yes. As I understand it we should next send a letter to Michelle Balinski, the editor of Mathematical Programming, with copies to editors of other journals of our field. Since we're a Committee of the Mathematical Programming Society, and since Balinski has already indicated a great desire to have us produce these guidelines, we can expect favorable treatment.

R. JACKSON: It's time now to sum up this session, since we have run out of time. I would like to point out that the next step for the Committee on Algorithms is to draft a set of proposed guidelines. We plan a panel discussion at the San Francisco meeting of ORSA in the spring of next year where these guidelines will be discussed by seven associate editors of the journals of our field. After that, a final version of the guidelines will be sent to the editors of those journals. The committee also is organizing a research exchange or newsletter to keep informed those persons who are interested in the work of the committee. We will be compiling a mailing list of "friends of the committee", and if anyone is interested please send your name and address to any member of the committee. With that, we can end the session by saying thank you all for coming.

CONTRIBUTORS

CROWDER, H.
IBM Research Center
Yorktown Heights, NY 10598

DEMBO, R. S.
Yale University
56 Hillhouse Avenue
New Haven, CT 06520

FILLIBEN, J. J.
Statistical Engineering Laboratory
National Bureau of Standards
Washington, DC 20234

GILSINN, J. F.
Applied Math Division
National Bureau of Standards
Washington, DC 20234

GREENBERG, H. J.
Chief, Supply & Integration Div.
Federal Energy Administration
Washington, DC 20461

GUTTERMAN, M. M.
Standard Oil Company of
Indiana
5049 Lee Street
Skokie, IL 60076

HELLERMAN, E.
System Software Division
Bureau of Census
Washington, DC 20233

KREUSER, J. L.
World Bank
1818 'H' Street, NW
Washington, DC 20433

MULVEY, J. M.
Harvard Business School
Harvard University
Soldiers Field Road
Boston, MA 02163

MYLANDER, W. C.
U. S. Naval Academy
Annapolis, MD 21402

O'NEILL, R. P.
Dept. of Computer Science
Louisiana State University
Baton Rouge, LA 70803

ORCHARD-HAYES, W.
International Institute for
Applied Systems Analysis
2361 Laxenburg
Austria

SHIER, D. R.
Applied Math Division
National Bureau of Standards
Washington, DC 20234

WILLIAMS, A. C.
Mobil Oil Corporation
Box 1025
Princeton, NJ 08540

WITZGALL, C.
Applied Math Division
National Bureau of Standards
Washington, DC 20234

154

LARGE SCALE MATHEMATICAL PROGRAMMING

(A TOTAL SYSTEMS APPROACH)

T. Prabhakar
Union Carbide Corporation
Building 82/507
Post Office Box 8004
South Charleston, West Virginia 25303

The modeling and solution of large-scale mathematical programming systems have been heavily influenced by the advent of large and high speed digital computers, powerful commercial software systems for mathematical programming, and special languages for matrix generation and report writing, and finally, by the increasing complexity of decision making in the business world. An attempt is made here to show how modeling and solution for large-scale business applications is approached nowadays from a total system view point starting from the problem definition and including the design of input and output systems, the formulation of the mathematical programming model, and the generation of financial and other business reports by drawing the information from the optimal solution and sensitivity analyses. A large linear programming application in use for production and distribution planning will be used for illustration.

A SEARCH ENUMERATION ALGORITHM FOR A MULTIPLANT, MULTIPRODUCT SCHEDULING PROBLEM*

Susumu Morito
Harvey M. Salkin

Department of Operations Research
Case Western Reserve University
Cleveland, Ohio 44106

CONTENTS

Contents
Abstract
1. Introduction
2. A Brief Description of the Production Process
3. Formulation of the Problem
4. The Algorithm
5. An Illustrative Example
6. Computational Experience
7. Conclusion
References

ABSTRACT

The concept of search enumeration in integer programming is applied to find the optimal schedule in a plastic injection industrial process. The production process involves molding parts using a molding machine which usually has several cavities. Each cavity accomodates a die which makes a single part. Given a set of customer orders which require certain production days (known), the problem is to find a schedule which minimizes the number of molding machine setups, while satisfying all technological and logistical constraints. This paper presents a search enumeration algorithm to find an optimal schedule. The algorithm is followed by an illustrative example. Some computational results are mentioned, and other comments concerning algorithm improvements are also given.

All work discussed here relates to part of an actual case study for a medium size Northeast Ohio Corporation. The entire case study resulted in the development of a computer system for production scheduling as well as for due date assignment, inventory control, machine allocation, and extensive data processing. In this article we concentrate our efforts on the optimization phase and its implementation for the production scheduling part of the system.

1. INTRODUCTION

The concept of search enumeration in integer programming is applied to find the optimal schedule in a plastic injection industrial process. The production process involves molding parts using a molding machine which usually has several cavities. Each cavity accomodates a die which makes a single part. Given a set of customer orders which require certain production days (known), the problem is to find a schedule which minimizes the number of molding machine setups, while satisfying all technological and logistical constraints. This paper presents a search enumeration algorithm to find an optimal schedule. The algorithm is followed by an illustrative example. Some computational results are mentioned, and other comments concerning algorithm improvements are also given.

All work discussed here relates to part of an actual case study for a medium size Northeast Ohio Corporation. The entire case study resulted in the development of a computer system for production scheduling as well as for due date assignment, inventory control, machine allocation, and extensive data processing. In this article we concentrate our efforts on the optimization phase and its implementation for the production scheduling part of the system.

2. A BRIEF DESCRIPTION OF THE PRODUCTION PROCESS

In this section, we describe the production process which involves molding plastic parts via a molding machine. A molding machine accomodates several (usually 6 or 8) dies, each of which corresponds to a specified part. Dies in a machine may be all different, but certain technological restrictions exist. Most importantly, these are die position constraints, and die length constraints. That is, due to their geometric characteristics and ease of removal from the die, certain parts must be produced by dies located in a mold position near the machine operator. These dies are said to require a "front position" in the mold, and are labelled "front runners." In contrast, the remaining dies can have either a front or rear position, where a rear position is at a farther distance from the operator. In addition to front or rear die positions, quality control dictates that some dies must be located closer to the center of the mold. Also, to avoid part breakage during their removal from the machine, the difference of lengths of adjacent dies can be no more than 3/4".

---

*No part of this document may be reproduced without explicit written permission of both authors.

It is undesirable and costly to stop the molding machine during a production run. Thus the molding operation is basically a continuous production process. New jobs to be processed correspond to orders for distinct parts. Based on an order quantity, the number of production days required is obtained. At the end of production for a certain job, a die must be pulled out of a cavity in the molding machine and a new die required for the next job is inserted. During this setup, the whole molding machine must be stopped, which means that production is lost. Although a setup can be completed fairly quickly, say in 30 minutes, it could take several hours until production stabilizes and acceptable parts are molded. Therefore, in order to maximize production, it is desired to minimize the number of setups.

Even though our actual problem involves multiple plants and multiple machines, as well as some other constraints (such as mold speed, plant specification, etc.), in order to simplify the discussion, we concentrate on a one-machine example. We also assume that orders correspond to distinct parts, only one die is available for each order and the scheduling period is known. (It is actually determined from a previous model in the system.) Our goal is to find a schedule, which minimizes the number of setups, while satisfying all technological constraints. The next section describes a search enumeration algorithm which finds an optimal schedule.

An Illustrative Example (1 machine, 11 orders)

Order Number	Production Days	Die Position	Die Length (inch)
1	2	Front	3.6
2	5	Rear	2.5
3	3	Front	3.2
4	7	Rear	2.8
5	3	Front	3.6
6	1	Front	3.3
7	3	Front	2.9
8	1	Rear	2.8
9	3	Front	3.0
10	4	Rear	3.0
11	4	Rear	2.7

The initial conditions of the machine schedule (usually due to the previous schedule) and an arbitrary schedule which satisfies the technological constraints is shown in Figure 1. In the figure, the shaded area indicates the initial conditions and the numbers in the shaded blocks are the die lengths. The scheduling period is assumed to be 5 days, and the search algorithm discussed in the next section seeks an optimal schedule which fills up this scheduling period with the smallest number of additional setups.

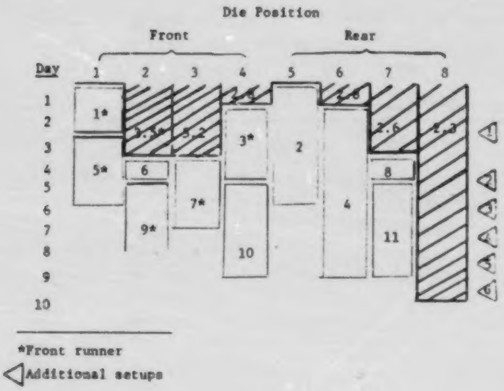


FIGURE 1. Initial Conditions (shaded area) & an Arbitrary Schedule for the Illustrative Example

3. FORMULATION OF THE PROBLEM

The algorithm is similar to a branch and bound approach used in scheduling theory (e.g., see Baker [1]), but the nature of the problem, as explained shortly, suggests a search enumeration used in integer programming (e.g., see Salkin [2]).* As in any enumeration algorithm, the process involves branching and bounding, and can be pictorially represented by a tree consisting of nodes and branches. The nodes correspond to subproblems and the branches link subproblems which differ by a single additional job fixed in a die position. We now discuss the procedure in context of an enumeration tree.

---

*A search enumeration, as opposed to a branch and bound enumeration, was used because of the following:

1) A branch and bound scheme tends to create many dangling nodes. That is, those that have not yet been considered for branching purposes. This may cause computer storage difficulties; whereas, a search enumeration deals with only one node at a time, at the cost of more bookkeeping, and does not require the storage of a set of dangling nodes (see, Salkin [2]). Saving computer storage, was especially important to us because the system is being implemented on a small computer (64K bytes of main core).
2) A clever bookkeeping scheme, based on the fact that a forward step corresponds to locating a die in a position with an earliest calendar opening, improves the performance of a search enumeration and uses a very minimal amount of storage.
3) A search enumeration always gives a current best feasible schedule and usually produces a near optimal schedule very quickly (see, Salkin [2]).

Let P^0, the initial node in the tree (see Figure 2), denote the problem containing n jobs. The problem P^0 can be partitioned into n subproblems, P^1_1, $P^1_2, \ldots, P^1_n$ by assigning jobs to the first opening (i.e., subsequent to forward steps). By "first opening" we mean the die position which has the earliest calendar opening. In case of ties, we can use any consistent rule such as the smallest index. Thus, P^1_1 is the original problem with job 1 fixed at the earliest opening; P^1_2 is the original problem with job 2 fixed at the earliest opening etc. In the Illustrative Example, the earliest opening is day 1 at positions 1 and 5. P^1_1 is then the original problem with job 1 assigned at position 1. (We are using the smallest index rule in case of ties.) Next, each of the subproblems can be further partitioned as in Figure 2. For instance, P^1_2 can be partitioned into P^2_{21}, $P^2_{23}, \cdots P^2_{2n}$. For example, in P^2_{21}, jobs 2 and 1 are assigned at the first two earliest openings in this order. In general, at level k, each subproblem contains k jobs already scheduled (or fixed in a die position). Each subproblem can be further partitioned into "at most" (n-k) subproblems, which form part of the level (k+1) subproblems. The reason why we say "at most" is because of the technological constraints mentioned earlier, which may limit the production of some parts at certain die positions. For example, problem P^2_{21} does not exist in the Illustrative Example as job 2, which is a front runner, cannot be placed in die position 5 which is a rear die position. This type of implicit enumeration may also be the result of the die length constraint.

We go down the tree until a schedule is completed and a solution is found. Then a backward step (a return to the previous subproblem) is taken and a regular search procedure starts. The detail of a search enumeration can be found in Salkin [2]. A part of a search tree for the Illustrative Example is given in the next section (Figure 3).

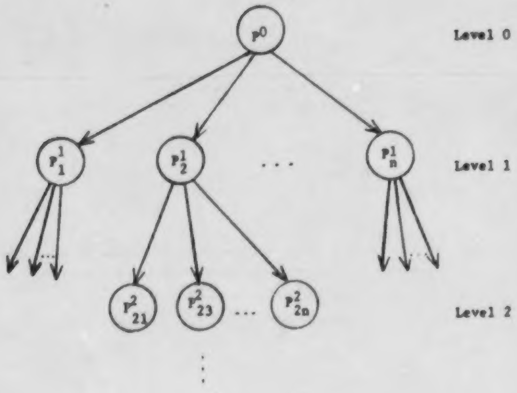


FIGURE 2. The General Enumeration Tree

4. THE ALGORITHM

Notation We denote the set S as a partial sequence of jobs from among the n jobs originally in the problem. For example, S = (2,4,3,1), means that job 2 is scheduled at the earliest opening, and job 4 at the 2nd earliest opening, job 3 at the 3rd earliest opening, and job 1 at the 4th earliest opening. Also, S7 = (2,4,3,1)7 means S = (2,4,3,1,7), or that job 7 is scheduled at the 5th earliest opening.

Algorithm Listing

Step 1 (Initialization)

Set Z*, the current smallest number of setups to an arbitrarily large value. Set k=0. The problem is P^0 and S is null. Go to Step 2.

Step 2

Find the (k+1)st earliest opening die position. In case of ties, select the smallest index. Go to Step 3.

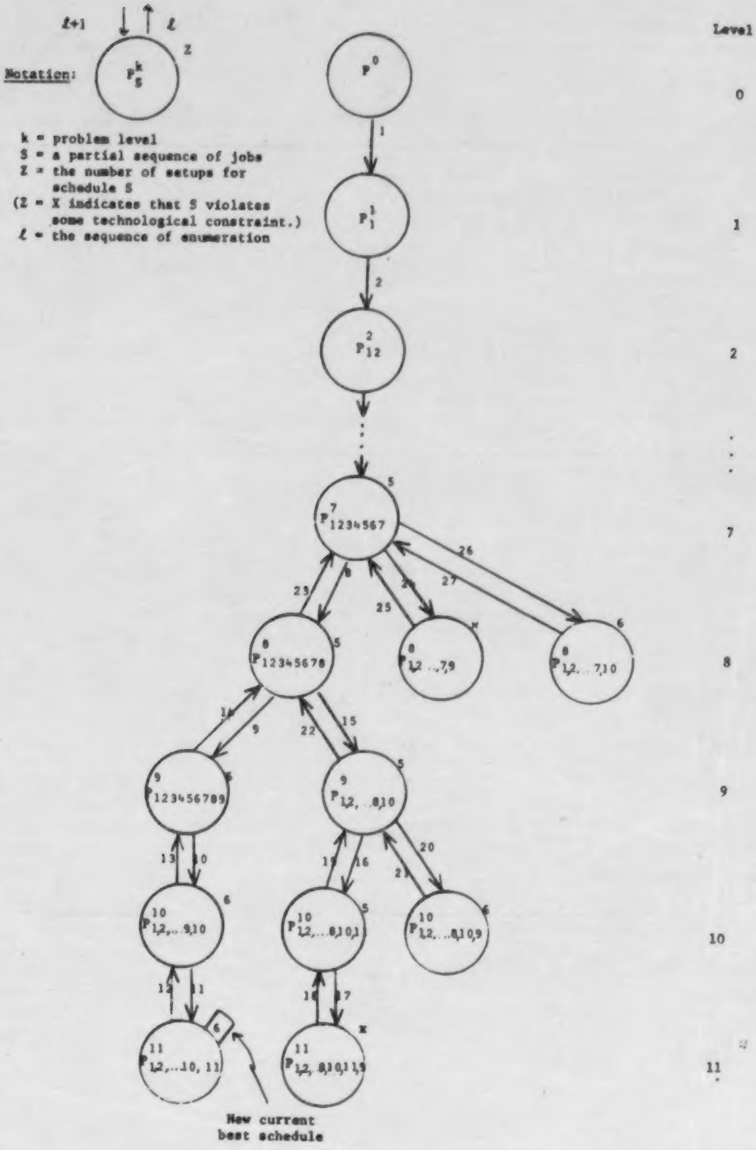
Step 3 (Forward Step)

At the (k+1)st earliest opening die position, try to schedule any one, say order i, of "untested" orders (i.e., exclude orders already tested or scheduled) which satisfy the die position (i.e., front/center/rear) and die length constraints. This defines problem P^{k+1}_{Si}.

(A) If there is no such order, go to Step 4 (Backward Step). Count the total number of setups thus far, denoted as Z.

(B) If $Z \geq Z^*$, mark the order as "tested" (at the current earliest opening die position), and repeat the Forward Step with another "untested" order.

(C) If $Z < Z^*$ and the schedule is completed, set $Z^*=Z$ (improved schedule found). Set k=k+1 and S=Si. Go to Step 4. (Backward Step).



Level
ℓ+1
ℓ
Z
Notation:
P^{k}_{S}
P^{0}
0
1
P^{1}_{1}
1
2
P^{2}_{12}
2
7
$P^{7}_{1234567}$
5
8
23
24
25
26
27
8
$P^{8}_{12345678}$
5
$P^{8}_{1,2,...,7,9}$
x
$P^{8}_{1,2,...,7,10}$
6
9
14
15
22
$P^{9}_{123456789}$
6
$P^{9}_{1,2,...,8,10}$
5
10
13
16
19
20
21
$P^{10}_{1,2,...,9,10}$
6
$P^{10}_{1,2,...,8,10,1}$
5
$P^{10}_{1,2,...,8,10,9}$
6
11
12
17
18
$P^{11}_{1,2,...,10,11}$
6
$P^{11}_{1,2,...,8,10,11,9}$
x
11

k = problem level
S = a partial sequence of jobs
Z = the number of setups for schedule S
(Z = X indicates that S violates some technological constraint.)
ℓ = the sequence of enumeration

New current best schedule

FIGURE 3. Part of the Search Enumeration Tree for the Illustrative Example

(D) If Z < Z* and the schedule is not yet completed (Forward Step), set k=k+1 and S=Si. Go to Step 2.

In any case, an order is never scheduled if its production duration exceeds the time the particular molding machine is expected to remain operable. (This is determined from a different model, not discussed in this article.)

Step 4 (Backward Step)

Back up to the last scheduled order, and mark the order scheduled at the die position being opened as "tested." In other words, if S=S'i (i.e., the last scheduled order is order i), remove order i from the current schedule and return to problem $P_{S'}^{k-1}$. Set k=k-1, and go to Step 5.

Step 5 (Termination Test)

If k < 0, stop; Z* is the minimum number of setups. Otherwise, count the number of setups, Z, for the current schedule (i.e., S'). If $Z \geq Z^*$, go to Step 4 and take another Backward Step. If Z < Z* (more precisely, if Z = Z*-1), go to Step 3.

5. AN ILLUSTRATIVE EXAMPLE

We now describe how the algorithm works using the Illustrative Example, and assume that initially an arbitrary schedule as shown in Figure 1 is given. This schedule is equivalent to P_S^{11}, where S = (1,2,3,4,5,6,7,8,9,10,11), with six (6) additional setups. (Note that setups due to the previous schedule are not counted.) Therefore, Z*=6, and the steps below follow. The tree corresponding to part of the computation is in Figure 3.

Step 4 (Backward Step)

Remove order 11 from the current schedule. Mark order 11 scheduled at the 11th earliest opening die position (i.e., die position 7 at day 4) as "tested." Set k=11-1=10.

Step 5 (Termination Test)

$k \nless 0$ and Z=6=Z*=6. Therefore, go to Step 4 and take one more Backward Step.

Step 4 (Backward Step)

Remove order 10 from the current schedule. Mark order 10 scheduled at the 10th earliest opening die position (i.e., die position 4 at day 4) as "tested." Set k=10-1=9.

Step 5 (Termination Test)

$k \nless 0$ and Z=6=Z*=6.

Step 4 (Backward Step)

Remove order 9 from the current schedule. Mark order 9 scheduled at the 9th earliest opening die position (i.e., position 2 at day 4) as "tested." Set k=9-1=8.

Step 5 (Termination Test)

$k \nless 0$ and Z=5≠Z*=6.

Step 3(D) (Forward Step)

Schedule order 10 at the 9th earliest opening die position. Z=5 < Z*=6, and k=8+1=9, and S = (1,2,3,...,8,10).

Step 2

The (k+1)st = 10th earliest opening die position is die position 4 at day 4.

Step 3(D) (Forward Step)

Schedule order 11 at the 10th earliest opening position. Z=5 < Z*=6, and k=9+1=10, and S = (1,2,3,...,8,10,11).

Step 2

The 11th earliest opening die position is die position 7 at day 4.

Step 3(B) (Forward Step)

Order 9, front runner, cannot be scheduled at the rear, and there is no other alternative.

Step 4 (Backward Step)

Remove order 11 from the current schedule. Mark order 11 scheduled at the 10th earliest opening as "tested." Set k=10-1=9 and S = (1,2,3,...,8,10).

Step 5 (Termination Test)

$k \nless 0$ and Z=5=Z*-1.

Step 3(B) (Forward Step)

Order 9 scheduled at the 10th earliest opening will make Z=6=Z*, and there is no other "untested" order at this opening.

Step 4 (Backward Step)

Remove order 10 from the current schedule. Mark order 10 scheduled at the 9th earliest opening as "tested." Set k=9-1=8 and S = (1,2,3,...,8).

Step 5 (Termination Test)

$k \nless 0$ and Z=5=Z*-1.

Then similar steps repeat by scheduling order 11 at the 9th earliest opening without producing a better schedule, and evenutally the enumeration reverts to level 8, level 7, etc. A part of the enumeration tree corresponding to the above description is given in Figure 3. After we check all alternatives at the 1st opening die position, we have the optimal solution, which is shown in Figure 4. Notice that Z*=3.

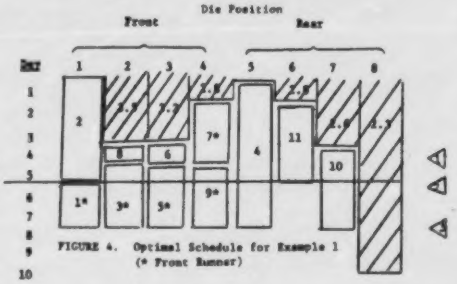


FIGURE 4. Optimal Schedule for Example 1
(* Front Runner)

6. COMPUTATIONAL EXPERIENCE

The algorithm has been coded in FORTRAN and is being implemented on a small IBM computer with 64K bytes of main core. A few smaller problems have thus far been tested successfully.

Additional computational behavior corresponding to the Illustrative Example (Section 5) is in Table 1. Running times relate to test runs on a UNIVAC 1108 computer.

TABLE 1
Computational Performance for Example 1

Level k	Time to reach level k first time after a Backward Step (seconds/Univac 1108)	Current smallest number of setups
11	0.000	6
10	0.001	6
9	0.005	6
8	0.011	6
7	0.015	6
6	0.030	5
5	0.034	5
4	0.060	5
3	0.147	4
2	1.290	4
1	1.509	3*
0 (Optimality)	5.984	3

*The optimal solution with 3 setups is obtained after 1.404 seconds

The algorithm can easily be extended to the multiple plant and multiple molding machine case. The actual case study involves many molding machines located at several plants. Each molding machine corresponds to a particular mold type and to a particular material* and so the algorithm may be applied directly to all molding machines with the same mold/material type. The multiple plants normally add certain technological, logistical, and/or quality control constraints. The resulting plant specification(s) naturally contributes to the algorithm's efficiency.

There are two different approaches when extending the algorithm to the multiple machine and multiple plant case. One approach lays out all machines in parallel and applies the algorithm directly. If we consider a two machine problem, each machine with eight cavities, the approach reduces to considering a one machine problem with sixteen cavities. Of course, the way to count the number of setups has to be modified, because a setup in one machine is independent of a setup in the other. On the other hand, a second approach to the multiple machine problem is to lay out all machines in series. In order to apply this approach, a planning horizon has to be introduced, where the idea is to schedule the machines during a given planning horizon. An order is scheduled only when it can start during the planning horizon. With this approach, the algorithm, during forward steps, schedules the complete planning horizon of the first machine, and then goes to the second machine, and so on. The backward step, works in an exactly opposite way. If this approach is adopted, then we are minimizing the number of setups which result from schedules that fill up the planning horizon. Therefore, it is possible that certain orders cannot be scheduled (or, more precisely, cannot be started) during the planning horizon in the optimal schedule (e.g., see Figure 6). At this time, the program uses the latter approach.

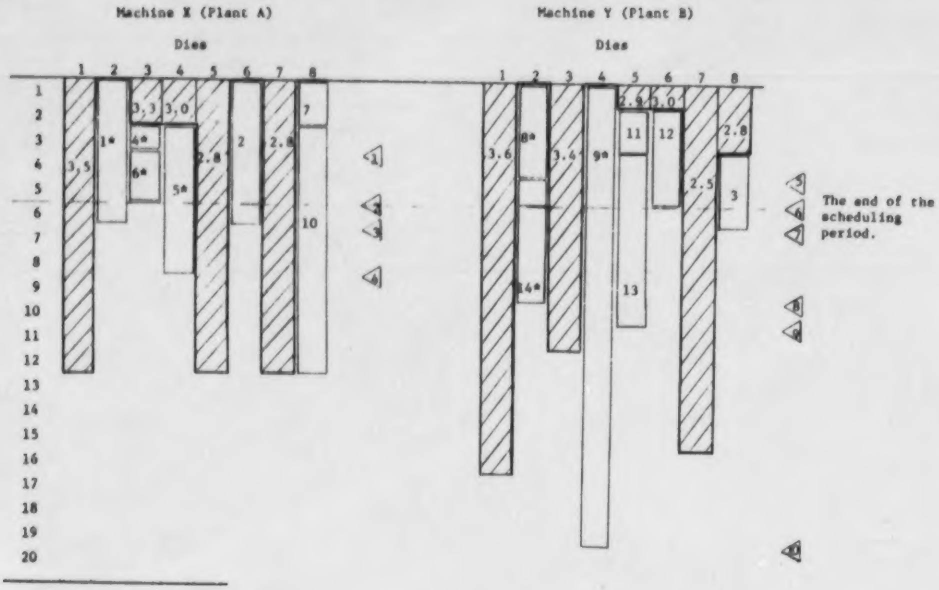
It should also be mentioned that the algorithm currently does not allow for "a hole" in a schedule. In other words, in a specific cavity of a molding machine, two subsequent orders must be scheduled in such a way that when the first one is finished, the other has to be started immediately subsequent to a set up. It is conceivable that by allowing a gap in a schedule (in practice, a cavity is blocked in this case) the number of setups can be reduced. Computations with a sample problem is below. In this example, we have 2 machines at two diffement locations, A and B. Due to certain logistical and other constraints not mentioned, it turns out that some orders must be produced at a specific plant, and thus we have a plant specification. Also, the two machines, denoted by X and Y have slightly different capabilities, and some orders must be processed by a particular machine (machine specification). If no specification is made, any order can be processed by any machine. Machine X is located at Plant A, and Machine Y at Plant B. Initial conditions of the machine schedule and an initial arbitrary schedule is given in Figure 5. The optimal schedule for the Example, found by the search enumeration, is shown in Figure 6. Notice that orders 6 and 8 are not scheduled during the current planning horizon.

EXAMPLE
(2 plants, 2 machines, 14 orders)

Order No.	Production Days	Plant Spec.	Machine Spec.	Die Position	Die Length
1	6	A	X	Front	3.0
2	6	A	X	Rear	2.5
3	3	-	-	Rear	2.4
4	1	-	-	Front	3.0
5	6	-	X	Front	3.0
6	2	A	-	Front	2.8
7	2	-	-	Rear	2.6
8	4	B	-	Front	3.3
9	19	B	-	Front	3.0
10	10	-	-	Rear	2.3
11	2	-	-	Rear	3.0
12	4	-	-	Rear	2.9
13	7	-	-	Rear	2.8
14	5	-	-	Front	3.5

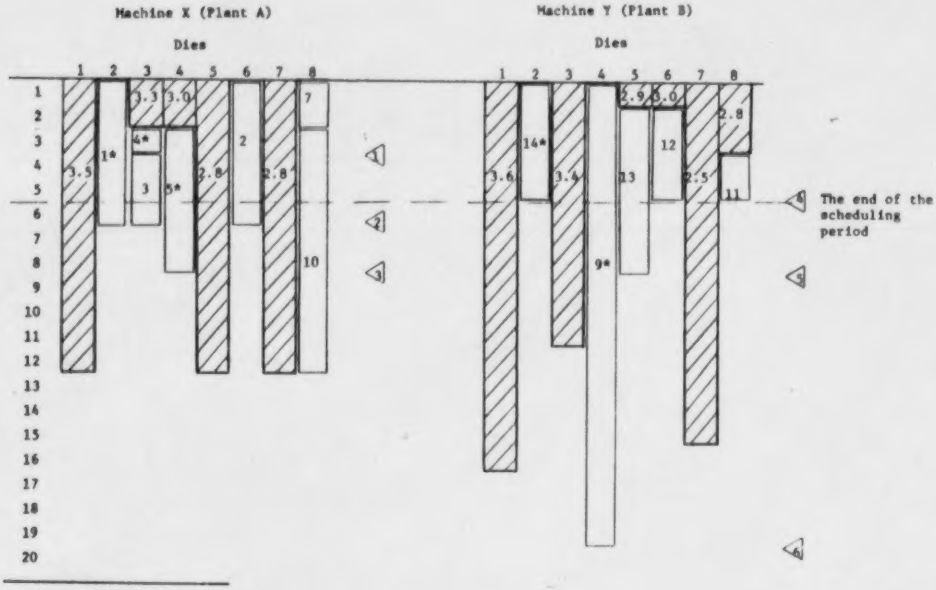
---

*The material specification is suggested in a previous model which is not discussed in this article. The final material specification is given by management.


Machine X (Plant A)
Dies
Machine Y (Plant B)
Dies
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
3.5
1*
3.3
3.0
4*
6*
5*
2.8
2
2.8
7
10
3.6
8*
14*
3.4
9*
2.9
3.0
11
12
13
2.5
2.8
3
The end of the scheduling period.

* Front runners

FIGURE 5. Initial Conditions (shaded area) and an Arbitrary Schedule for the Example


Machine X (Plant A)
Dies
Machine Y (Plant B)
Dies
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
3.5
1*
3.3
3.0
4*
3
5*
2.8
2
2.8
7
10
3.6
14*
3.4
9*
2.9
3.0
13
12
2.5
2.8
11
The end of the scheduling period

(Orders 6 and 8 not scheduled)

FIGURE 6. Optimal Schedule for the Example

The model is being tested for problems with 5 plants and about 30 molding machines. Fortunately, as there are several material types and mold types, it can be subdivided into smaller problems.

Even though the current model assumes deterministic production days, in reality, some orders have a range of production days such as either 7 or 8 days. This allows more flexibility in the scheduling phase, and may further reduce the number of setups. The model has been expanded to incorporate variable production days. (This is examined at each step in the enumeration and thus the schedules may not be optimal in a global sense.)

In general, algorithm trys at most $n!$ combinations, where n is the number of orders. However, in computations, implicit enumeration substantially reduced the number of combinations that must be examined. Judging from the fact that a fairly good schedule is often obtained early in the enumerative process and also that a substantial amount of computations is usually spent to show that the current best solution is, in fact, optimal, heuristics which will curtail the computations are now being considered. For example, we can restart the algorithm by changing the earlier part of the schedule drastically when the computational progress slows, and eventually terminate all computations.

7. CONCLUSION

The problem of finding an optimal schedule which minimizes the number of setups in a plastic molding operation is formulated and solved by search enumeration. Even though the size of the problems tested so far are relatively small (at most 15 orders), the original problem combined with all technological, logistical, and other constraints yields several smaller problems. Each of these with a clever heuristic rule, that generates a good, if not optimal, schedule within a reasonable amount of computer time, should be solved quickly using the algorithm. It is currently in an extensive testing stage for use as a subroutine in a production scheduling and inventory control computer system.

REFERENCES

[1] Baker, K.R., Introduction to Sequencing and Scheduling, John Wiley, New York, 1974.

[2] Salkin, H.M., Integer Programming, Addison-Wesley, Reading, Mass., 1975.

AN IMS-GAMMA 3 DATABASE EDITOR

E. B. Brunner
Gulf Oil Corporation

INTRODUCTION

As personnel from all levels of today's industries are becoming more involved with the "man-computer dialogue", we see the phenomenon occurring that more non-computer trained people are utilizing its facilities. For many years now, the shift has been away from having highly trained computer people provide the services to execute a program. Increasingly, the developed program is turned over to the user to allow data tabulation and execution to be done by him. This, in turn, has caused a need for an easy method for the user to do this data tabulation and program execution.

One feature that can help to accomplish this is the use of a datbase. A database being defined as "a collection of interrelated data stored together with controlled redundancy to serve one or more applications in an optimal fashion; the data are stored so that they are independent of programs which use the data; a common and controlled approach is used in adding new data and modifying and retrieving existing data within the data base."[1]

Utilizing this concept with its "common and controlled approach to add and modify data" we can devise programs by which a user can tabulate data and execute his program easily.

Furthermore, this man-machine interface should fulfill the following objectives:

1. Meet the needs of the user.
2. Understandability of results.
3. Reliability of execution.
4. Ease of use.
5. Timely results.

EVOLUTION OF A PROJECT

Early in 1973, a request was made by the Gulf Oil Trading Company for an analytical tool (i.e., L. P. model) that would determine the relative value of crude oils for different refinery situations. The model was developed in card deck form using the matrix generator-report writer product of Bonner & Moore Software Systems, GAMMA 3. The model preformed very satisfactorily and gave clearly understood results. The familiar complaint then came from the customer that the mechanics of tabulating the information on keypunch forms, waiting for and checking the keypunching and then submitting the deck was too time consuming and prohibited them from obtaining the needed results within a half-day or less. What could we do about this? The solution to the problem seemed to be some form of interactive setup to eliminate the need for the card deck. This requirement turned out to be two-fold, a method or program to create and maintain an input database and then a way of submitting a job stream. For their purposes we had met items numbered 1, 2 and 3 of our objectives list; but as yet, had not addressed items 4 and 5.

Being that IBM's Information Management System (IMS) was already resident at Gulf and provided the needed database capabilities via a Cathode Ray Tube or batch program, and that the project sponsor's analysts were familiar with using the CRT for financial work, it was decided to use IMS to meet these requirements.

THE DATABASE EDITOR

The database and its programs were to handle data in either the table or list format utilized in GAMMA 3, and in our design each table and list was to be a database record. Figure 1 is an example of a GAMMA 3 table and list in card image form. The table in database record structure would appear as in Figure 2.

Further, since it was projected that many models would probably use this database, a two-character prefix was defined to be used as a means of distinguishing unique sets of tables and lists. Each table/list name would be prefixed by these two characters to reserve it for a particular user. Additionally, password security was to be made available for each user-chosen prefix. The IMS non-conversational program to manipulate these records was written PL1 and called GAMED.

---

[1] Martin, James, "Principles of Data-base Management," Prentice Hall Inc.

Presently, it is running on an IBM 370/158 under VS and is accessible through a 3270 CRT or equivalent.

As displayed on the next two figures (3 and 4), GAMED provides capabilities to display and edit database tables and lists. For display purposes the user can show:

1. A table with its text and/or data.
2. A list or list with text.
3. A table or list row names.
4. Table column names.
5. A tabulation of tables and lists in a database set.

For editing the user can:

1. Modify
    - A. Table/list text
    - B. Table data
    - C. Row/column maximum counts
2. Add
    - A. Complete tables/lists
    - B. Table rows and/or columns
    - C. List rows
3. Delete
    - A. Complete tables/lists
    - B. Table rows and/or columns
    - C. List rows
    - D. Entire database set
4. Duplicate
    - A. Individual tables/lists
    - B. Entire database set
5. Replace Row and Column Names

Since the structure of a table/list seems to have the two distinct parts of text and data, displays are made with the table/list text formatted on a screen in a separate manner from table data. (Figures 5 and 6.)

Upon initialization of the transaction GAMED, the user obtains the display in Figure 7. As explained in the directions, the user's two-character prefix is to be placed within the parentheses, the table/list name that is to be acted upon replaces the word NAME, the field with the word TEXT specifies the current format in use, the three letter designator following the character string 'FUNC:' is the function field (SEE), the space between the words SEE and PAGE is the options fields, and the PAGE clause on the right has indicators for multiple page messages.

If we use the prefix (EB), table name SAMPLE, function SEE (Figure 8) and depress the ENTER key, the resultant display will look like Figures 9 and 10. Note that this is a database display of the GAMMA 3 table listed previously and Figure 9 is the text portion and Figure 10 is the data.

If we would like to modify some data entry on Figure 10, the function is changed to MOD, and we move the cursor down to the value to be modified and change it. In this case, it is the value at the intersection of row R2 and column C3 (Figure 11). Then depress the ENTER key and receive the response in Figure 12.

Once GAMED was completed, there still remained the problem of having the GAMMA 3 model generator access these database tables and lists. This was accomplished by two additional programs, called GAMLOAD and GAMUNLD. GAMLOAD will take a specially formatted file of tables and lists created by GAMMA and load them onto the database under a specified prefix, whereas GAMUNLD will unload all or selected tables of one or more prefixes and create this specially formatted file to be readily useable by GAMMA. In addition, another utility program exists to list and/or punch this file in card image form.

After these programs were completed, the first part of the original two-fold user problem was solved; now programs existed to maintain the input database and make it available to the model. This eliminated any need for data cards since the analyst running the model could enter the data via the CRT.

While the effort to produce GAMED was taking place, a second IMS program called JOBEDIT was developed. Its purpose was to maintain and edit a database containing jobstream card images. The database, called JOBFILE, was separated into members designated by an eight-character name. As an added feature, any of these members that began with a job card could be submitted as a job stream with a RUN command. (Example, Figure 13.) This, of course, satisfied our second user requirement.

The customer was then provided with a member on JOBFILE that contained a job stream to execute the model. After the data tables and lists were updated for the current problem, the user would RUN his job stream and get results within the desired time frame of a half-day or less.

We now have 17 production L. P. models and three non-L. P. model applications being run in this manner by users at various geographical locations throughout the country. This type of operation has been so successful that we now receive requests from our user companies that stipulate that their models must be developed to be run from an IMS terminal.

Finally, I would like to list some other IMS capabilities that we have added to our operations. As part of a run, the user can optionally:

1. Receive reports at a CRT and have these available for recall.

2. Receive reports as hard copy at a local printer.
3. Load onto the database, tables/lists created during execution.

Figure 14 shows a diagram of the overall operation from the interaction of GAMED to the unload optimize and reload cycle, thereby given what we feel is a complete and unified system.

```
TABLE SAMPLE ,T=10 ,R=3 ,C=3
     *  '    T     '    C1        C2        C3
     R1  RED           1.0       2.0       3.0
     R2  BLUE          4.0       5.0       6.0
     R3  ORANGE        7.0       8.0       9.0
LIST (COLORS) ,T=12
      RED   'PRIMARY     '
      BLUE  'PRIMARY     '
      GRAY  'ACHROMATIC  '
      BROWN 'SECONDARY   '
      BLACK 'ACHROMATIC  '
      GREEN 'SECONDARY   '
      ORANGE'SECONDARY   '
      TAN   'SECONDARY   '
```

Figure 1

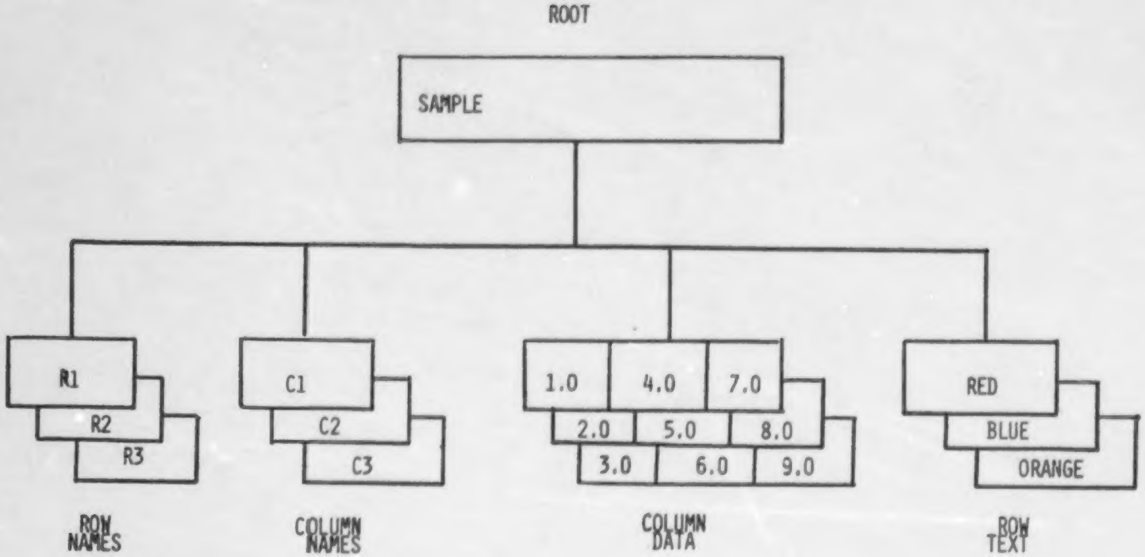

ROOT
SAMPLE
R1
R2
R3
C1
C2
C3
1.0
4.0
7.0
2.0
5.0
8.0
3.0
6.0
9.0
RED
BLUE
ORANGE
ROW NAMES
COLUMN NAMES
COLUMN DATA
ROW TEXT

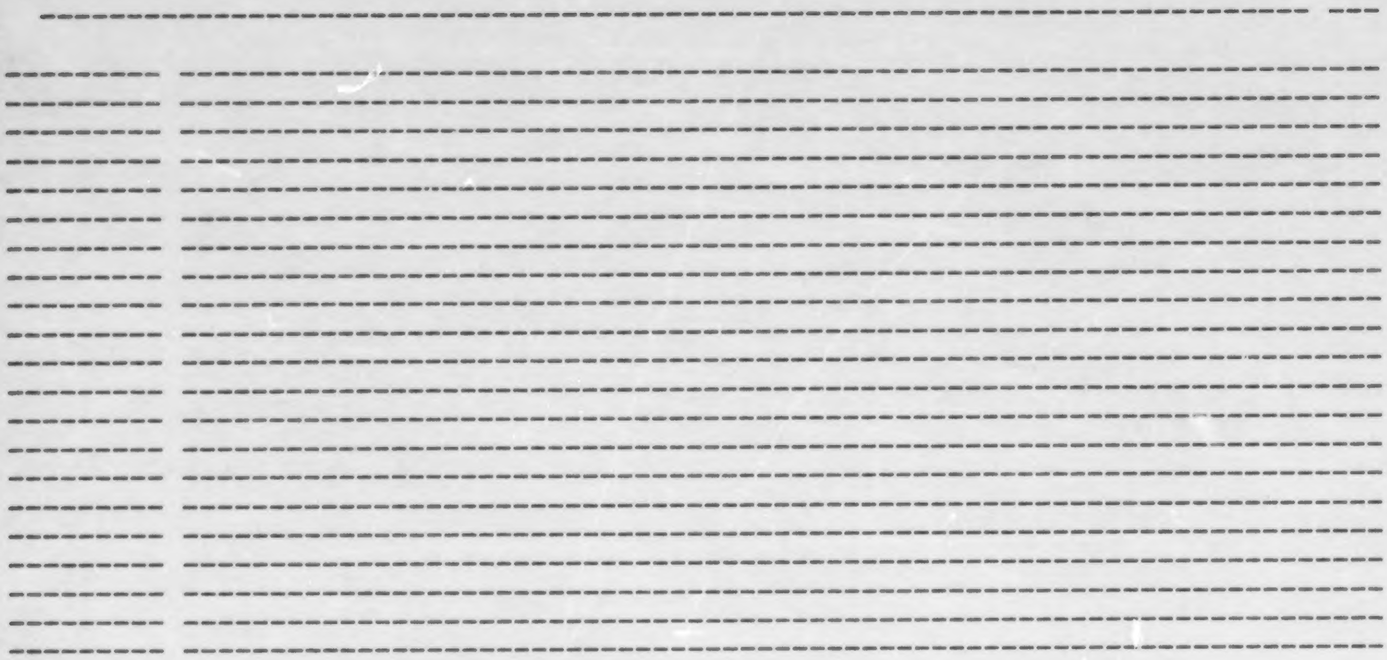
Figure 2

DISPLAY CAPABILITIES:
    A. TABLE
        1. TEXT
        2. DATA
    B. LIST
*
    C. TABLE OR LIST ROW NAMES
*
    D. TABLE COLUMN NAMES
*
    E. TABLES AND LISTS IN A DATABASE SET

Figure 3

EDIT CAPABILITIES:
    A. MODIFY
        1. TEXT
        2. DATA
*
    B. ADD
1. COMPLETE TABLES/LISTS
        2. TABLE ROWS AND/OR COLUMNS
        3. LIST ROWS
*
    C. DELETE
        1. COMPLETE TABLES/LISTS
        2. TABLE ROWS AND/OR COLUMNS
        3. LIST ROWS
        4. DATABASE SET
*
    D. DUPLICATE
        1. INDIVIDUAL TABLES/LISTS
        2. COMPLETE DATABASE SET
*
    E. REPLACE ROW AND COLUMN NAME

Figure 4



Text Format

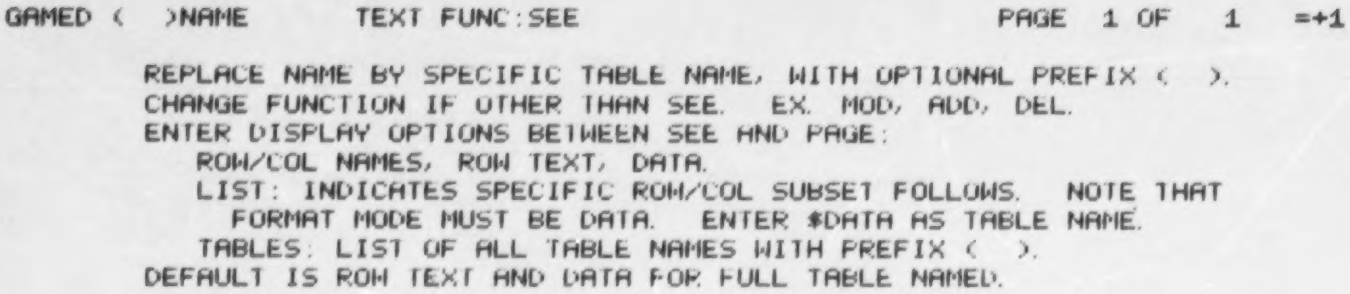
Figure 5

```
  ------------------------------------------------------------------------ ---

--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
--------- ------------- ------------- ------------- ------------- -------------
```

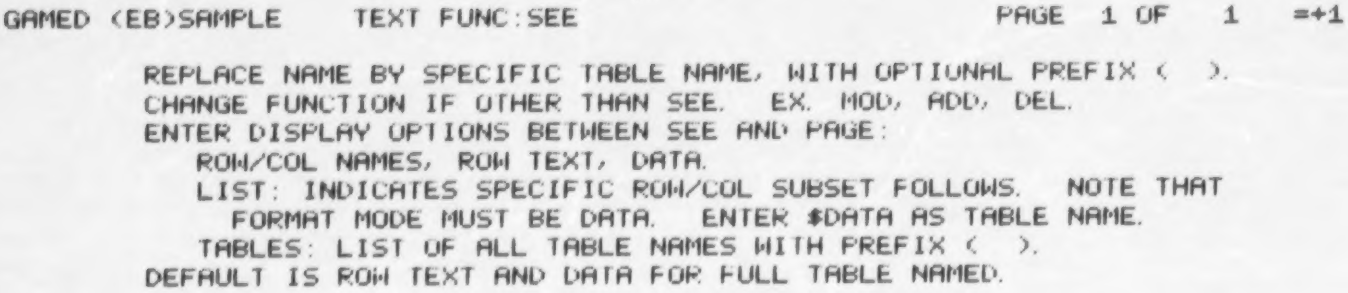
Data Format

Figure 6



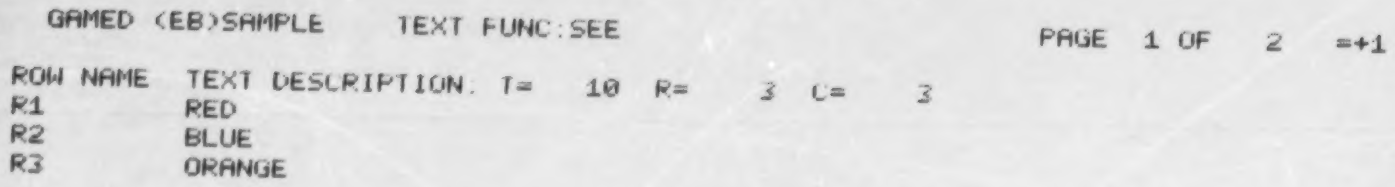
```
GAMED (  )NAME       TEXT FUNC:SEE                           PAGE  1 OF   1   =+1
          REPLACE NAME BY SPECIFIC TABLE NAME, WITH OPTIONAL PREFIX (  ).
          CHANGE FUNCTION IF OTHER THAN SEE.  EX. MOD, ADD, DEL.
          ENTER DISPLAY OPTIONS BETWEEN SEE AND PAGE:
             ROW/COL NAMES, ROW TEXT, DATA.
             LIST: INDICATES SPECIFIC ROW/COL SUBSET FOLLOWS.  NOTE THAT
               FORMAT MODE MUST BE DATA.  ENTER $DATA AS TABLE NAME.
             TABLES: LIST OF ALL TABLE NAMES WITH PREFIX (  ).
          DEFAULT IS ROW TEXT AND DATA FOR FULL TABLE NAMED.
```

Figure 7



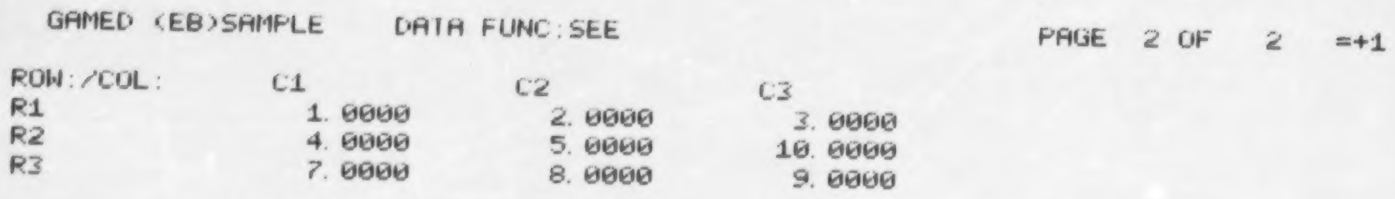
```
GAMED (EB)SAMPLE    TEXT FUNC:SEE                           PAGE  1 OF   1   =+1
          REPLACE NAME BY SPECIFIC TABLE NAME, WITH OPTIONAL PREFIX (  ).
          CHANGE FUNCTION IF OTHER THAN SEE.  EX. MOD, ADD, DEL.
          ENTER DISPLAY OPTIONS BETWEEN SEE AND PAGE:
             ROW/COL NAMES, ROW TEXT, DATA.
             LIST: INDICATES SPECIFIC ROW/COL SUBSET FOLLOWS.  NOTE THAT
               FORMAT MODE MUST BE DATA.  ENTER $DATA AS TABLE NAME.
             TABLES: LIST OF ALL TABLE NAMES WITH PREFIX (  ).
          DEFAULT IS ROW TEXT AND DATA FOR FULL TABLE NAMED.
```

Figure 8



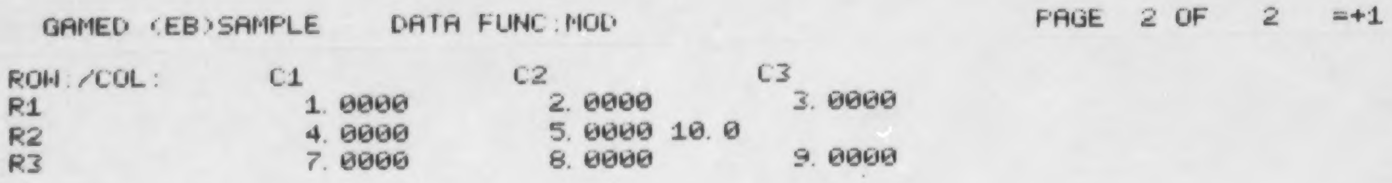
```
  GAMED (EB)SAMPLE     TEXT FUNC:SEE                                  PAGE  1 OF   2   =+1
ROW NAME  TEXT DESCRIPTION. T=   10  R=    3  C=    3
R1        RED
R2        BLUE
R3        ORANGE
```

Figure 9



```
  GAMED (EB)SAMPLE     DATA FUNC:SEE                                  PAGE  2 OF   2   =+1
ROW:/COL:        C1             C2             C3
R1                 1.0000         2.0000         3.0000
R2                 4.0000         5.0000        10.0000
R3                 7.0000         8.0000         9.0000
```

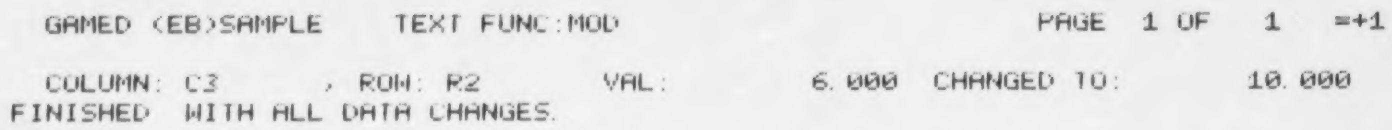
Figure 10



```
 GAMED (EB)SAMPLE    DATA FUNC:MOD                          PAGE  2 OF   2   =+1

ROW:/COL:       C1             C2             C3
R1                 1.0000         2.0000         3.0000
R2                 4.0000         5.0000 10.0
R3                 7.0000         8.0000         9.0000
```

Figure 11



```
 GAMED (EB)SAMPLE    TEXT FUNC:MOD                          PAGE  1 OF   1   =+1

 COLUMN: C3      , ROW: R2      VAL:         6.000  CHANGED TO:        10.000
FINISHED  WITH ALL DATA CHANGES.
```

Figure 12

```
//MSPGMEBB JOB 106010000004002,'BRUNNER X334',MSGLEVEL=1                      1
//*MAIN ORG=RM070,CLASS=SVS                                                    2
//  EXEC PGM=IEHPROGM                                                         20
//SYSPRINT DD SYSUUT=A                                                        30
//DD1 DD UNIT=DSK11,VOL=SER=GSYS40,DISP=SHR                                   40
//SYSIN DD *                                                                  60
    SCRATCH VOL=DSK11=GSYS40,DSNAME=MSPG.GAMED.PREFIX                         70
    SCRATCH VOL=DSK11=GSYS40,DSNAME=MSPG.GAMED.PREFIXES                       80
    SCRATCH VOL=DSK11=GSYS40,DSNAME=MSPG.GAMED.PREFIXS                        90
    SCRATCH VOL=DSK11=GSYS40,DSNAME=MSPG.GAMED.PREF                          100
/*                                                                           110
REQUESTED MEMBER DISPLAYED=MSHA*PGM
```

Figure 13

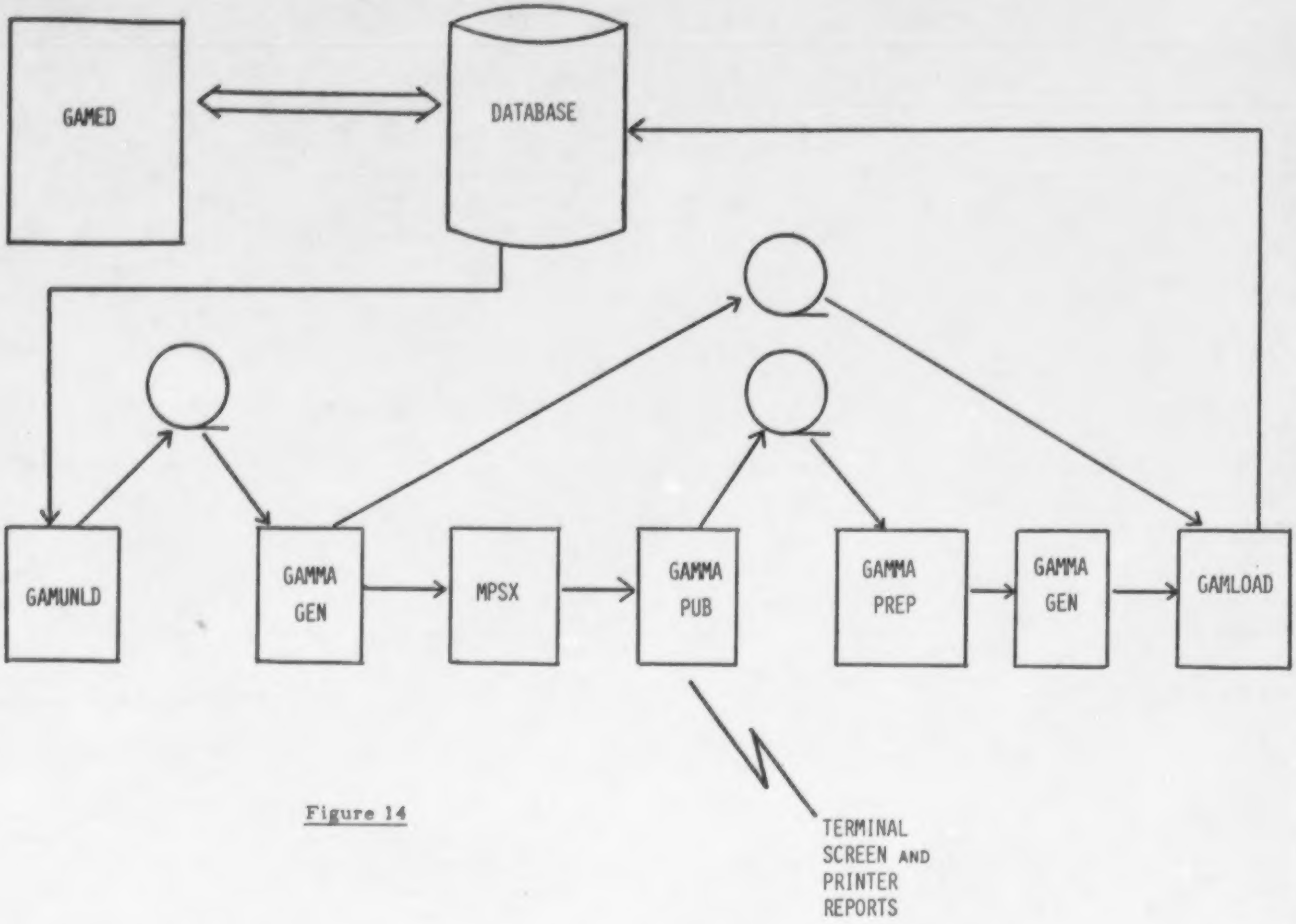

GAMED
DATABASE
GAMUNLD
GAMMA GEN
MPSX
GAMMA PUB
GAMMA PREP
GAMMA GEN
GAMLOAD
TERMINAL SCREEN AND PRINTER REPORTS

Figure 14

SOFTWARE TOOLS FOR COMBINING LINEAR PROGRAMMING WITH ECONOMETRIC MODELS*

Michael J. Harrison

COMPUTER RESEARCH CENTER FOR ECONOMICS AND MANAGEMENT SCIENCE
National Bureau of Economic Research, Inc.
575 Technology Square
Cambridge, Massachusetts 02139

Abstract

Mathematical programming is increasingly being combined with other mathematical modeling techniques. A combination which has proved particularly fruitful in the field of energy modeling is that of an econometric model and a linear program. Some practical experience has shown that although such combined models give rise to unexpectedly few mathematical difficulties during solution, the computer software which is currently available is very inconvenient for implementing these models. This paper analyses the sources of these difficulties, and describes some new software tools which are being developed to aid in the construction of combined modeling systems.

1. Introduction - The State of the Union

Recently, mathematical models which have as major components both an econometric part and a linear programming (LP) part have come into vogue. Hogan [1] has described how such a model was constructed and solved at the Federal Energy Administration for the Project Independence Evaluation System. Jorgenson [2] has proposed a combination of the Brookhaven Energy LP model and the DRI interindustry econometric model. Shapiro [3] describes these and similar models, and discusses mathematical approaches to their solution. Currently such solution methods require iteration between the econometric model and the LP; the econometric model provides supply and demand data, and the LP returns an optimal distribution of commodities to meet the demand and the associated shadow prices. The process terminates when an equilibrium set of prices is attained. This paper addresses the question of what software tools are needed by a person who wishes to develop and implement an algorithm of this nature with the minimum computer programming effort.

It will be useful at the outset to distinguish between two separate but connected aspects of the problem of combining LP and econometrics on the computer. One aspect is that of data base management; that is, the marshalling of the original raw data into a form suitable for computation, and the production of meaningful output. The other is that of system integration; that is, the provision of mechanisms to control the execution of the processes, and to facilitate transfer of data across boundaries between systems. This paper is principally concerned with system integration; data base management, LP model generation, and report writing form the principal themes of companion papers [5], and so these topics will not be treated in detail here.

Let us first examine the problems that confront a modeler who tries to use existing software. Hogan [4] has described the difficulties that arose during the implementation of the PIES model [1]. Despite there being no rigorous theoretical foundation to the algorithm, its convergence was found to be unexpectedly rapid (6 to 10 iterations being required). Hogan was fortunate in that the integrating portion of the model could be readily coded in FORTRAN, and that APEX, the LP optimizer he used, has the most convenient interface with FORTRAN of any commercially available large-scale LP code. He has stated [4] that the major difficulty in the project was that of matrix generation, but it would probably be more accurate to say that the real problems were in data base management, as will be clear from a full account of the project [6]. Nevertheless, although the combination of the two models did not constitute the greatest difficulty in the project, it is clear that the integration process was by no means simple and straightforward.

The paucity of published results on the successful combination of modeling techniques tends to corroborate the view that implementation of such combined models is difficult in practice. Apart from Hogan's work, we are forced to draw on our own experience at NBER for illustration of the difficulties which are encountered. The major stumbling block is that the econometric system and the LP system, having grown up separately, do not have an easy way to exchange data; furthermore, each has its own execution control mechanisms and neither will allow itself to be subservient to the other. As an example, consider a pilot implementation of the least absolute residuals algorithm for regression, in which the data were set up and manipulated in TROLL [7] and the minimization performed using the SESAME LP system [8]. In view of the experimental nature of the project, it was decided to use an existing LP system rather than to code one from scratch, even though there are modified versions of the simplex algorithm known which will solve this particular problem more efficiently than a standard simplex code. Since TROLL and SESAME must each reside in a different virtual machine (because TROLL contains its own virtual machine supervisor), the only means for passing data between them was as spooled card-image files. In addition, the user had to control the whole process manually from the console because there was no way to pass control between the two virtual machines. This latter difficulty could have been overcome, at the cost of considerable system programming effort, by arranging that each virtual machine could suspend itself and activate the other at appropriate times. In the end, this project was abandoned because the effort involved seemed too great for the expected benefits. TROLL and SESAME were both designed to be flexible systems for experimental rather than production use, and the problem of combining their facilities was not expected to be so difficult.

---

*This report has not undergone the review accorded official NBER publications; in particular, it has not yet been submitted for approval by the Board of Directors.

Lest it be thought that NBER's systems are the only offenders, consider the way IBM approached a similar problem, that of combining APL with MPSX [9]. Again, data are communicated by means of card-image files. So is program control information -- the APL program constructs a card file of MPSX control language commands and passes this file to the MPSX compiler. This mechanism is both clumsy and inefficient, but is necessary because MPSX accepts input of control information only in the specific form of card images.

The APL/MPSX combination, though achieved by a roundabout means and with no little programming effort, was successful enough to warrant publication. Naturally, failed attempts to adapt an existing LP system to a new environment do not reach the literature, but the author is aware from personal contacts with people in the modeling world that many such attempts have been envisioned. Because of the obvious difficulties which would ensue, few such attempts proceed beyond the speculative stage.

We may conclude, then, that the state of the union is not good.

2. Limitations of Current Software

Let us summarize the main points arising from the general overview given in Section 1. The areas in which current software exhibits deficiences are as follows:

(a) Data base management and generation of LP models from raw data

(b) Control of LP procedures by other programs

(c) Transfer of data between LP procedures and other programs

Category (a) is treated in companion papers [5] and will not be discussed here. Categories (b) and (c) overlap slightly (for example, control information might be regarded as data passed to the LP program), but they are essentially separate.

A major drawback of current large-scale LP-solving programs (with the partial exception of APEX) is that they cannot be invoked from another program written in a standard language such as FORTRAN or PL/I by a subroutine call or similar mechanism. APEX allows its constituent routines to be called from a FORTRAN program once the general APEX environment has been established. Other LP systems can be invoked only by means of a special language peculiar to each system. One reason for this may be that most operating systems do not provide a suitable dynamic loading capability; LP codes are now so voluminous that they are too big to include in a load module with other programs of comparable size. However, the main reason appears to be that LP programs are deemed to require such a specialized environment for efficient operation that only the LP system itself can be entrusted with creating this environment, and that therefore the LP system must be in overall control of the whole process. Some progress seems to have been made in modifying commercial large-scale LP systems to provide a more convenient interface for the experimental user; IBM claims that MPSX/370 has the required flexibility [10]. However, it is clear that the only way to achieve a really clean control interface is to design the LP code with that requirement in mind at the outset, rather than to try to adapt an existing code. This may entail the sacrifice of some raw computational efficiency in the interests of greater flexibility of application.

Transfer of problem specification data into LP systems is still essentially at the BCD stage. Beale has championed a "card image" data interface on the grounds that such a medium provides the maximum standardization and flexibility [11]. Whilst it is true that such an interface is desirable to facilitate transfer of data between computers of different manufacturers, card images are a poor means of passing data between programs running on the same machine. However, Beale's philosophy has prevailed thus far, and many current matrix generator programs, such as GAMMA [12] and MaGen [13], produce as their output a specification of the LP problem in a card image format, usually that of MPS/360 which has become a sort of informal standard. Such a mode of operation does have the advantage that a single matrix generator can produce output which is suitable for input to a variety of different LP systems, but the intermediate data form was originally designed for human readability and does not lend itself readily to manipulation by the computer. Indeed, some of the complexity and inefficiency of matrix generators can be attributed to exactly this cause.

Some matrix generators, notably DATAFORM [14] and DATAMAT [8], produce a specification of the problem directly on a special file, referred to variously as the "problem file" or "models file". This allows the LP to dispense with its INPUT or CONVERT routine which it would normally use to convert the card deck into an internal representation. This representation also facilitates a direct revision of the model, which would otherwise have to be performed by altering the matrix generator program, or its data, or both, and executing the matrix generator again. The data on the problem file are, however, accessible to the user only via special routines which themselves depend on the environment set up by the LP program.

In a similar way, output of solution information from typical LP systems is at the BCD level. In order to be read by another program, such data has to be stored on a file in a format which is compatible with the other program, and unit records of 80 characters or so are a common choice. Some LP systems allow the option of filing the solution data in some internal form on a "results file" and provide system routines by which the user may access the data. This is neater, but has the drawback that these routines cannot be used outside of the control scope of the LP program.

Current LP programs fail to distinguish sharply enough between two sorts of data about the LP model -- that pertaining to the structure of the model (its size and the pattern of non-zero elements), and the actual values of the coefficients. As explained in Section 1, algorithms for solving combined models typically use the results of one LP run to revise the input data for the next. In such revision, only the coefficients are changed and not the structure of the model. Hence, it is advantageous to keep these data entities separate, and to make the coefficient values easily accessible to the caller of the LP code. Some modern LP codes have made partial provision for this in the shape of "indirect" coefficients -- that is, coefficients whose values are initially specified as character-string names which are later bound to numerical values. However, the mechanisms provided for altering the values associated with particular names have not been especially convenient.

In all of the foregoing, it is clear that the greatest hindrance to combining a modern large-scale LP system with an econometric system of comparable complexity is the insistence of the LP system on setting up its own environment and control mechanisms. In particular, operations on the LP data base are usually only possible within the environment set up by the LP or matrix generator. The external interface of a typical modern large scale LP system has been designed to be convenient for a human; such an interface is not well adapted for use by a computer program. DATAFORM ostensibly has the capability of solving non-linear problems by an iterative algorithm involving dynamic revision and solution of successive linear programs. However, MPS III must remain in overall control of the process, and although it is possible to invoke from DATAFORM subroutines written in, for example, FORTRAN, it would not be possible to invoke another computer system, such as a simulator.

3. New Methodology

Interactive Computing

We make the assumption, in considering new approaches to dealing with the problems outlined above, that computers will be used increasingly in an interactive mode rather than in a batch mode, especially during the development phase of a programming project. Although interactive operation has little effect, in principle, on the computer systems required for successfully combining LP with other models, it greatly affects the flavor of our approach. It is assumed here that the reader has had some exposure to interactive computing, and appreciates both the advantages it affords the user and the difficulties it entails for the system designer.

NBER Support Systems

The Computer Research Center of NBER has implemented and is extending a set of system facilities designed, among other things, to aid the developer of new modeling systems. The evolution of the philosophy behind these systems has been explained previously [15]. The Applications Control System (ACOS) provides a shared hierarchical file system, a supervisor, an input/output manager for I/O not connected with the file system, and the Applications Control Language (ACOL). ACOL is a language for writing programs which interpret commands entered from a terminal according to a programmer-defined syntax, and for controlling the execution of program modules in response to such commands. For more detailed information on ACOS and ACOL the appropriate manuals [16] should be consulted.

ACOS supports, as a subsystem, DASEL, which provides both a language for specifying mathematical models and a language in which to implement algorithms [17]. The modeling language, like that in TROLL [7] allows equations to be entered symbolically, but it goes beyond TROLL in offering facilities for symbolic manipulation, such as differentiation. DASEL has a library procedure which will solve small LP problems, and also allows more powerful LP programs to be called by means of its interlanguage communication feature. ACOS also supports XMP, a flexible subsystem which solves large-scale LP models.

Analysis of the Model Combination Problem

In what follows, we restrict our attention to one particular combination of models -- that required to solve the equilibrium problem [1,3]. This will make the discussion clearer and more concrete, and the principles developed will be applicable to almost any combination of modeling techniques. The steps in the solution of the equilibrium problem are:

```
1. Define the econometric model (EM).
2. Define the LP model.
3. Specify an initial solution of prices
   for the EM.
4. DO while convergence criterion not met:
      a) Compute demands from EM.
      b) Pass demands to LP.
      c) Solve LP to compute optimal
         allocation
      d) Retrieve shadow prices from LP.
      e) Test for convergence; if not
         converged, compute revised prices
   END DO
5. Display results.
```

It is readily apparent that we require the EM program, or some program hierarchically superior to both the EM program and the LP program, to be in control of the overall execution of this algorithm. The LP program acts as a subroutine, albeit an intricate one with complex input and output. Let us now consider each of the above steps in turn to see how they can be implemented with the right software.

Definition of the EM

We assume that the econometric modeling language system has a means of allowing the model to be specified by means of symbolic equations.

Definition of the LP

Small LP models can be defined in terms of constraint expressions and an objective with only minor extensions to the EM modeling language. One needs to be able to attach labels to constraints, so that the objective may be identified and so that constraints may be referred to subsequently, and the language must permit the operators "<=" and ">=" in addition to "=". Of course, the LP program must be able to accept the model in such a form, rather than in a BCD representation; however, this requirement is easily met, since all the processing to do with parsing the equations is already present in the EM system. Both the DASEL library procedure and XMP can accept an LP model in this form.

Formulation of a large LP constraint by constraint would be impossibly cumbersome. We have, therefore, developed a language called XML which allows an LP to be specified as constraints indexed over sets, and which incorporates the Σ operator [5]. An LP model may be expressed in XMP in essentially the same way as it would in purely mathematical terms, with some concessions to the shortcomings of computer typography. XML allows coefficient values to be specified symbolically; the actual numerical values may be stored in a data structure (an n-dimensional array) in the user's file system, and a binding process associates the XML symbolic name with the file name of the data structure. This is somewhat analogous to the use of indirect coefficients in some current LP systems, but it is more flexible, as will become clear in the subsequent discussion of model revision.

Specification of an initial solution

The precise details of this will depend on the EM system, but any good system will make construction of a vector of values a simple process.

Main Interative Loop

If the language of the EM system allows the LP program to be invoked by subroutine call or similar mechanism, and if it also has constructs for testing, branching, and looping, this iterative loop can be controlled by the EM system. Both TROLL and DASEL provide the appropriate constructs, but only DASEL has LP as a callable function.

We can assume that steps 4a and 4c are achieved simply by invoking the respective programs, and that the EM system provides the facilities for computing the revised priced in step 4e.

The passing of data in steps 4b and 4d is the real crux of the problem. It is easily solved if the LP is small enough to be expressible in the EM language; for example, if DASEL were used, the data could be passed simply as DASEL variables. Specifically, the demands are DASEL variables to the EM program, which assigns values to them, although they act as constants to the LP program, for which they are coefficients in the right-hand-side (or bound values). The LP returns the shadow prices (π-vector) as an explicit vector which is also a DASEL variable, and therefore the EM program can retrieve values from it. The difficulty with this approach is that it requires the user to keep track of the elements in these vectors by numerical index values; this is satisfactory for a small LP model, but impractical for a large one.

If one attempts to extend this approach to larger LP models, other problems arise than keeping track of variables by index number. Even with a computer which provides virtual storage, the available "core" storage may be exceeded, because the code and data for both the EM and the LP exist simultaneously. Further, the LP program constructs its working data from the modeling-language description of the LP from scratch each time; for a small LP this is a reasonable approach, but it would clearly be grossly inefficient to generate a large LP problem each time from an XML specification when the only changes concern a few right-hand-side values. Finally, when retrieving solution values we wish to refer to rows and variables in terms of the XML model specification, and use this as a selection mechanism, because the full primal and dual solution to a large LP would occupy unnecessarily a good deal of memory if it were expressed as unpacked vectors.

Efficient Revision of LP Models

In order to solve the problem of lack of space we require a supervisory program which passes control alternately to the EM program and the LP program. In this way, much of the data areas and executable code may by overlaid. ACOL provides just such a capability.

Unfortunately, if we remove control from the EM program and demand an efficient mechanism for making small revisions to the LP model, we encounter a problem with the scope of the data. This arises as follows. To solve an XML model, the user supplies the data (as n-dimensional arrays), specifies a binding of XML identifiers to the data, and invokes XMP with the name of the XML model as an argument. At this level of operation, data entities such as "right-hand-side" which are internal to the LP system are not accessible, and so values in them cannot be changed. It would be possible to allow access to these internal data if either they were preserved in core between calls to the LP code (ACOL permits this), or if the data were saved on the file system, but both these approaches are open to objection.

The good solution to the problem lies in taking advantage of the hierarchical structure of XMP, which was designed to facilitate just such use. The top-level procedure XLPSOLVE consists of calls to the second-level procedures GETPROB (which converts the problem from XML form to internal form) and SOLVE (which performs simplex iterations). Internal data are accessible at this level, and so

180

small revisions to the right-hand-side values, for example, are easy; system routines are provided for this, in order to maintain modularity in the Parnas [18] sense. The particular right-hand-side elements being changed may be referred to by the name of the corresponding constraint expression, so that the user need not be concerned with numerical indices. The conventions governing the use of names are explained in the section "Retrieval of Results". We can then rewrite Step 2 of the algorithm on page 3 as:

2. a. Define the LP in XML.
   b. Invoke GETPROB to convert to internal form.

and Step 4c as:

   c. Invoke SOLVE.

Observe that GETPROB and SOLVE can be invoked independently of XLPSOLVE, something which is not possible with current commercial LP systems. It is possible in XMP because data entities are passed as arguments, and therefore no routine is required to set up a special environment in which these procedures operate.

Retrieval of LP Results

The particular problem we are considering requires only the retrieval of a few shadow prices. In general, one may assume that the solution values most often requested will be primal and dual values, and that tableau values will be required much less frequently; hence, XMP makes primal and dual values especially accessible. The internal form of the final basis is also made accessible so that it can be saved and used to initiate a future execution of SOLVE.

As with revision of a model, the scope of data presents a problem with retrieval of results. However, we can overcome the problem by using a special feature of ACOL. When simplex iterations have finished, SOLVE makes a recursive call to ACOL (recursive because ACOL invoked SOLVE in the in the first place). SOLVE is still active and therefore all its internal data are preserved. In response to the recursive call, the ACOL interpreter starts to read from whatever environment invoked SOLVE. In this particular case, SOLVE was invoked from an ACOL program, and so more of this program is read. The statements that are read are requests to SOLVE for solution information which are implemented by the ACOL program; it calls entry points in the SOLVE program which return solution values. The solution values are then passed via ACOL to the EM program. There is an explicit ACOL request to unwind the recursion and leave the SOLVE environment.

When retrieving solution information, the user must be able to refer to the constraints and variables of the LP model by meaningful names. With conventional matrix generator systems this is conceptually easy, since all these systems require that the user specify these names precisely in the form of 6- or 8-character concatenations. Normally, these names will have been generated in some regular fashion which allows selection of the desired solution information to be achieved by a masked-name matching technique. However, XML has no concept of names of this type; its names refer to whole classes of constraints and variables, with individual members of a class corresponding to particular values of elements of indexing sets.

The style of naming adopted by XML is very convenient for selecting classes of elements of the model. In the particular case with which we are concerned, suppose the set of demand constraints is named DEMAND and is indexed over regions of the country and over type of energy; these index sets might be "NE","MA",...,"NW" (for New England, Middle Atlantic,...,North West) and "EL","NG,...,"RO" (for electricity, natural gas,...,residual oil), respectively. Then the whole class of constraints is referred to by the name DEMAND; cross-sections would be denoted by, for example, DEMAND("NE",) or DEMAND(,"NG"); and a single constraint by a name such as DEMAND("NW","EL"). The ACOL request to retrieve the shadow prices for the demands might be something like:

```
PI ("DEMANDS","PRICES")
```

where PI is the request. The quotes denote that each argument is a literal character string; PRICES is the name of a DASEL variable in which the values of the shadow prices will be returned. The values are returned in the same order as the members of the corresponding index sets, and are then available to the EM program for further computation.

Reporting of Final Results

Since the LP program is essentially a subroutine in the whole process, the EM program has the primary responsibility for reporting the final results. Both TROLL and DASEL have extensive facilities for manipulating data and displaying data in both tabular form and as plots. It is apparent, however, that the result retrieval operations discussed above form a partial set of primitives for the construction of an LP report writing system.

4. Summary and Conclusions

Although the discussion in Section 3 is largely centered around a particular model combination, the principles expounded admit of ready generalization, and the software described is capable of dealing with many other possible combinations in a straightforward manner. The main points to notice are as follows:

(a) ACOS provides a uniform file system, thus eliminating one common source of incompatibility, and a dynamic loader.

(b) ACOL allows one to write code (in PL/I, FORTRAN or whatever) which may be invoked either from the terminal or from another program without the code having to distinguish between the two. Thus the algorithmic modules of a system such as XMP and their associated ACOL driver programs are readily incorporated into systems. Also, ACOL allows error conditions to be treated in a uniform manner and at the appropriate level.

(c) DASEL provides a suitable language in which to specify econometric models, and algorithmic procedures for operations such as regression and simulation. It also has facilities for maintaining a data base, manipulating data, and displaying data.

(d) XML is a convenient language for specifying large-scale LP models, and XMP is a subroutine capable of solving such models.

Our view is that extremely flexible applications software results if carefully structured libraries of procedures are available to be connected together as a user desires under the general environment of a suitably tailored operating system.

5. Acknowledgements

The author is indebted to Bob Fourer, Roy Marsten, and Bill Northup for helpful discussions; to Bill Hogan and John Pearson for documentation on the Project Independence studies; and to Ed Kuh for explaining what an econometric model is.

6. References

[1] W.W. Hogan, "Project Independence Evaluation System Integrating Model", Office of Quantitative Methods, Federal Energy Administration, 1974.

[2] D. Jorgenson, "An Integrated Reference Energy System and Interindustry Model for the U.S. Economy", pp. 211-221 in *Notes on a Workshop on Energy Systems Modeling*, Technical Report SOL75-6, Systems Optimization Laboratory, Stanford University, 1975.

[3] J.F. Shapiro, "Steepest Edge Decomposition Methods for Mathematical Programming/Economic Equilibrium Planning Models", *Working Paper OR 046-76*, Operations Research Center, Massachusetts Institute of Technology, 1976.

[4] W.W. Hogan, "Large Models can be More Difficult to Generate than to Solve", *Discussion Paper* prepared for Computer Science and Model Building Conference, Vail, Colorado, 1975; "Matrix Generation, Report Writing, and the Scale of Future Applications", working paper, Fall Meeting ORSA/TIMS, Miami, FL, November 1976.

[5] R.H. Fourer, "XML -- A Modeling Language for Mathematical Programming." Preliminary Report. NBER Computer Research Center, December 1975.

[6] *Project Independence Report*, Federal Energy Administration, Washington, D.C., GPO No. 4118-00029, November 1974.

[7] *TROLL: An Introduction and Demonstration*, NBER Computer Research Center, 1975, and references therein.

[8] *SESAME: Design and Capabilities Overview*, NBER Computer Research Center, 1974, and references therein.

[9] R.S. Goncharsky, A. Rauch, and W.W. White, "Large Scale Mathematical Programming in an APL Environment", *IBM Philadelphia Scientific Center Technical Report 320-3027*, 1973.

[10] "MPSX/370 Design Objectives", IBM Corporation, 1974.

[11] E.M.L. Beale, "Matrix Generators and Output Analyzers", *Proceedings of the Princeton Symposium on Mathematical Programming*, ed. H.W. Kuhn, Princeton University Press, 1970, p. 25.

[12] GAMMA 3 User Manual, Bonner & Moore Associates, Inc., Houston, TX, 1973.

[13] MaGen/PDS User Manual, Haverly Systems, Inc., Denville, NJ, 1973.

[14] DATAFORM User Manual, Management Science Systems, Rockville, MD, 1970.

[15] M. Eisner and R.W. Hill, "Topologically Speaking, Time-Sharing Systems in Time Spiral; or, How a TROLL Got DASELed", *Proceedings of Computer Science and Statistics: 8th Annual Symposium on the Interface*, Health Sciences Computing Facility, University of California, Los Angeles, 1975, p. 27.

[16] *ACOS Overview, publication D0082; ACOL Reference Manual, publication D0085*, NBER Computer Research Center, 1975.

[17] *DASEL User's Guide, publication D0084*, NBER Computer Research Center, 1975.

[18] D.L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules", *Comm. ACM, 15* (1972), 1053.

Data Base Management Techniques for Mathematical Programming

R.H. Bonczek, C.W. Holsapple, A.B. Whinston
Purdue University

Supported in part by OWRT Grant # 6538-62-1310

ABSTRACT

This paper describes the fundamental concepts of data base management and proceeds to suggest the utility of these concepts for the data handling aspects of mathematical programming. A general network-based data management system is used to illustrate data structure definition, data manipulation and non-procedural query capabilities with respect to math programming. The emphasis is upon flexibility and convenience for both the implementors and users of mathematical programming algorithms.

Implementation of mathematical programming algorithms entails the storage and manipulation of large volumes of data. Our principal objective is to suggest ways in which both the implementors and users of mathematical programming may benefit from developments in the growing field of data base management. For implementors, data base management can provide powerful data storage mechanisms and facile data manipulation capabilities. The storage mechanisms are powerful with respect to their capacities to handle complex data relationships and both numeric and non-numeric data within a single data structure. The data manipulation capabilities allow access to the data without concern for overlaying or management of auxiliary storage devices. For users, data base management can provide enhanced data maintenance facilities including data editing and problem formulation.

We commence with a review of important data base terminology and concepts. These are utilized to illustrate specific ways in which various issues of mathematical programming may be treated with data base management techniques. So, the emphasis is on data handling rather than mathematical or computational details. Naturally, the development of an algorithm will exploit the the available data structure and data handling capabilities. As will be noted shortly there are several varieties of data bases; the network variety, which is the most flexible and general, will be used for illustrative purposes. A language is presented which allows full manipulative capabilities, with respect to data organized according to network structures. This language is completely independent of the types of data that are stored and it may be used within the confines of such commonplace languages as FORTRAN, ALGOL, or COBOL. An example is provided demonstrating how this data manipulation language can be used to accomplish the pivot operation. Finally, a high-level, English-like, non-procedural query language is described. The query language is independent of the data structure and automatically interfaces application routines with desired data.

DATA BASE CONCEPTS

A data base has two major attributes: its logical structure and a collection of data values which are stored according to that structure. The logical structure (also called the schema) is effectively a blueprint describing what types of data values may be included in the data base and how each of these types is related to the other types. The schema serves as the basis for all data manipulation; addition, deletion, retrieval, or modification of a data value is accomplished by references to its corresponding type in the logical structure.

Every data base structure can be described in terms of three basic features: data item types, record types, and sets. A particular schema is composed of data item types which are related to one another by record types and by sets. So data item types are considered to be the most fundamental "building blocks" of a logical structure. Each data item type is identified by a unique name and indicates a distinct type of data. For instance each of the data item types VARIABLE-ID, VARIABLE-DESCRIPTION, CONSTRAINT-ID, CONSTRAINT-DESCRIPTION, COEFFICIENT-VALUE, and COEFFICIENT-SOURCE indicates a distinct type of data which we may want to include in a data base. Each data item type represents many occurrences of data values of that type within the data base. It is important to understand that a data item type is a component of the schema, whereas a data value is an actual instance of data of some type. For example, the data item type VARIABLE-ID may represent the data value occurrences 'X1' through 'X100'.

A logical structure is formed by relating the various data item types with each other. There are two varieties of relationship: aggregation and association. A record type is a named aggregate of data item types. For example, the record type VARIABLE may be composed of the data item types VARIABLE-ID and VARIABLE-DESCRIPTION. Figure 1a gives a pictorial representation of this, where the record type is shown as a rectangle labeled VARIA-

BLE and enclosing the names of its data item types. Just as a record type is an aggregate of data item types, an occurrence of a record type is an aggregate of data value occurrences (i.e., one data value occurrence for each of the data item types which constitute the record type). A sample occurrence of the record type VARIABLE is 'X1' and 'AMOUNT OF RESOURCE 5 TO BE USED.'

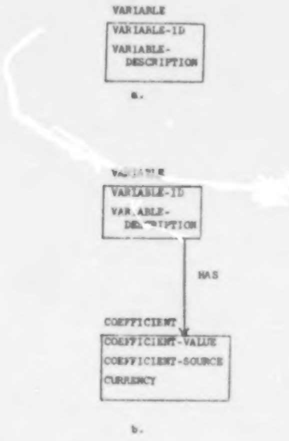


Figure 1. Example of Structural Components of a Data Base

The second kind of relationship allows record types (and therefore item types) to be associated with each other by means of a set relation. This notion of a set was popularized by the CODASYL Data Base Task Group (DBTG) Report of 1971 [1]. The DBTG notion of a set must not be confused with the concept of a mathematical set; there is no direct relation between the two. A set is described in terms of an owner record type and a member record type such that there is a one-to-many relationship between each occurrence of the owner type and occurrences of the member record type. But although there may be many (or one or zero) member occurrences associated with each owner occurrence, for a given set a particular member occurrence may be associated with no more than one occurrence of the owner record type. Consider the record types COEFFICIENT and VARIABLE shown in Figure 1b, where the former is an aggregation of such data item types as COEFFICIENT-VALUE and COEFFICIENT-SOURCE. The two reocrd types are related via the set named HAS. This set is denoted by an arrow pointing from its owner record type (VARIABLE) to its member record type (COEFFICIENT). Recalling the definition of a set, we observe that each occurrence of VARIABLE may have many occurrences of COEFFICIENT associated with it, but a given occurrence of COEFFICIENT may be associated with no more than one occurrence of VARIABLE. Thus the set HAS properly describes the relation between variables and coefficients in the context of linear programming. If we reverse the direction of the arrow in Figure 1b, then the structure no longer supports the relation between coefficients and variables which is inherent in linear programming. Thus an important issue in data base management is the design of a schema that correctly reflects the relationships among item types.

Not only does a set provide information about relationships among occurrences of its owner and member record types, but it also permits the member occurrences associated with each owner occurrence to be logically ordered (for purposes of storage and access) according to some criterion. For example, given an owner occurrence of the set HAS, its member occurrences may be ordered in an ascending fashion according to the values of their respective COEFFICIENT-VALUE occurrences.

Figure 2 displays an example of record occurrences organized according to the schema of Figure 1b. Three expressions are shown having five variables and a total of twelve coefficients. In the diagram beneath the expressions, circles denote record occurrences of the type shown in the right margin. The arrows show which member occurrences are owned by each owner occurrence via the set indicated in the right margin. For purposes of diagrammatic clarity, record occurrence details are not shown; e.g. no data values of VARIABLE-DESCRIPTION are included in occurrences of the VARIABLE record type. Also notice that no zero coefficients are stored. The diagram clearly displays the idea of a set. Each owner record type may be associated with many member record occurrences, but no member occurrence is associated with more than one owner occurrence.

Expression

$E1 = 100X1 + 50X2 + 30X3 - 70X4 - 30X5$

$E2 = - 10X2 + 13X3 - 10X4 - 90X5$

$E3 = 12X1 + 3X4 + 10X5$

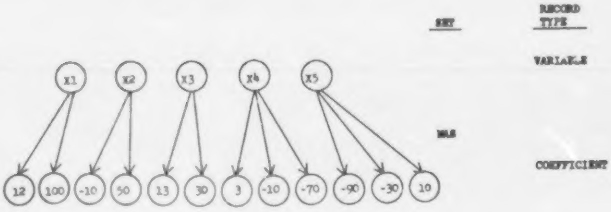


Figure 2. An Occurrence Level Example Based on the Schema of Figure 1b.

It is readily apparent that the structure of Figures 1b and 2 does not distinguish between coefficients of one expression and those of another. In a subsequent section, we deomonstrate a manner in which this shortcoming can be remedied.

A TAXONOMY OF DATA BASE STRUCTURES

Utilizing the concepts and terminology introduced in the preceding section, we can illustrate the varieties of data base structures which can exist. The most elementary data structure is composed of a single record type. This is analogous to a FORTRAN array. A slightly more complex circumstance occurs when data values are organized into a number of disjoint (i.e., no set relationships) record types; this corresponds somewhat to a group of FORTRAN arrays. Figure 3a shows a linear structure; this means that each record type is the owner of at most one set and is the member of no more than one set. Figure 1b is an example of a linear structure. A data base may be composed of several disjoint linear structures. A more flexible structure, the tree, is depicted in Figure 3b. Observe that in this type of structure a record type may be the owner of more than one set, but it can be the member of at most one. Thus there is a unique path between any two record types. Figure 3c shows a network data structure. This is the most general of all data structures that are describable in terms of the three features: data item type, record type and set. That is, the record type, linear structure and tree are all special cases of a network.

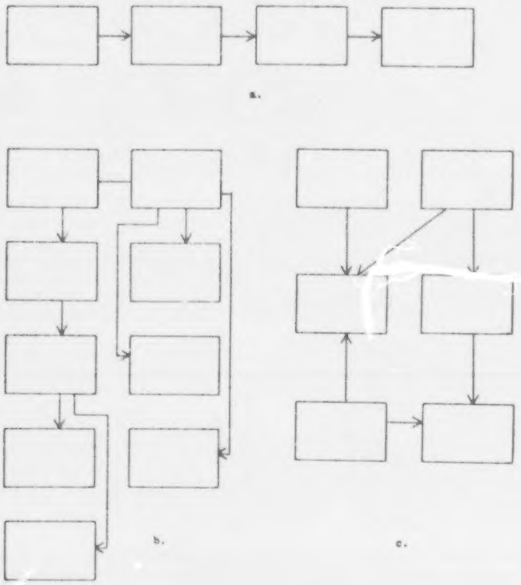


Figure 3. Varieties of Data Strucutre

Since a record type in a network may be the member of many sets and also the owner of many sets, multiple paths may exist between two record types. The term path indicates a sequence of sets which are related via record types, such that any two record types on the path are related by a unique subsequence of the path's sets(i.e., the path contains no loops). It must be emphasized that although a path is composed of directed arcs, the definition of a path contains no reference to direction. The direction of arcs (sets) within a path offers no impediment to the traversal of record types. Figure 4a presents a special type of path which may exist in a network structure. The path between VARIABLE and EXPRESSION consists of the two sets HAS and CONTAINS, related through the record type COEFFICIENT. A path of this sort is used to indicate a many-to-many relationship between variables and expressions. That is, a variable participates in many expressions and an expression contains many variables. But each occurrence of COEFFICIENT is 'owned' by one occurrence of VARIABLE (via the set HAS) and by one occurrence of EXPRESSION (via the set CONTAINS). It, therefore, relates a variable and an expression without violating the definition of a set. This is exemplified in Figure 4b, using the expressions of Figure 2.

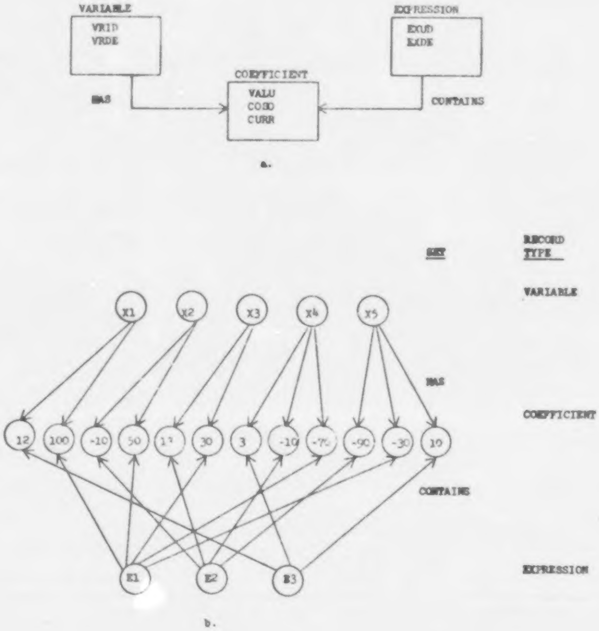


Figure 4. Example of a Many-to-Many Relationship

LANGUAGE FOR DATA BASE DEFINITION

We preface this section with a few words of caution. It cannot be asserted too strongly that the diagrams in Figures 1 through 4 are not flow charts or PERT diagrams; they can in no way be considered to represent algorithms. As already stated, the diagrams with rectangles depict logical structures according to which data may be organized and a diagram with circles gives specific examples of data values which have been organized on the basis of some schema. The diagrams of logical structures provide a simple mechanism for representing and communicating about a data base. However, they are by no means simplistic, but rather quite powerful. This power arises from the capacity of a logical structure to support the storage and access of data values, in such a way that a data base user need not be explictly concerned with the complicating factors of physical storage and access. For instance, the particular data base management system that is examined in this paper is implemented in terms of doubly linked lists which are stored on a direct access device. The physical storage and retrieval of data for this system requires extensive pointer manipulation and the buffering of blocks of data from the direct access device. However, the typical data base user is concerned with none of this; physical manipulations are automatically accomplished by submitting requests in terms of the logical structure.

The logical structure of a data base is formally defined via a Data Description Language (DDL). We illustrate this by using the DDL of a particular data management system: the Generalized Planning System(GPLAN) developed at Purdue University. The DDL for the data structure of Figure 4a is shown in Figure 5. In this version of DDL all identifiers are restricted to four characters in length. As the figure indicates, the DDL is non-procedural. Each record type is followed by the item types which compose it. Each item type is defined in terms of its name, its type (e.g. integer, character, real) and its size. The size of a character item type is given by its number of characters; other size specifications are in terms of computer words. The DDL description of a schema serves as input to a DDL Analyzer which produces schema tables. These tables are subsequently used by the Data Manipulation Language (DML) for creating and accessing record occurrences.

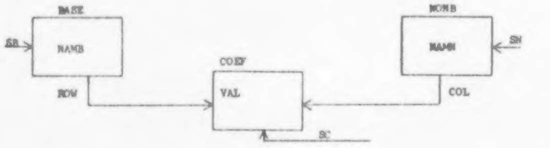
		type	size	comments
RECORD	VAR			
ITEM	VRID	INTEG	1	VARIABLE-ID
ITEM	VRDE	CHAR	20	VARIABLE-DESCRIPTION
RECORD	EXPR			
ITEM	EXID	INTEG	1	EXPRESSION-ID
ITEM	EXDE	CHAR	20	EXPRESSION-DESCRIPTION
RECORD	COEF			
ITEM	COSO	CHAR	16	COEFFICIENT-SOURCE
ITEM	CURR	CHAR	4	CURRENCY
ITEM	VALU	REAL	1	VALUE
SET	MAS			
OWNER	VAR			
MEMBER	COEF			
SET	CON			
OWNER	EXPR			
MEMBER	COEF			

Figure 5. Example of Schema Specified in Terms of DDL

DATA MANIPULATION LANGUAGE

The GPLAN Data Manipulation Language [2], [3] consists of approximately seventy commands which allow the data base user to perform the following types of functions: 1) open and close the data base; 2) create and delete record occurrences; 3) set currency indicators; 4) add and remove record occurrences from sets; 5) store and retrieve data from record occurrences; and 6) search through sets for particular record occurrences. In the GPLAN implementation each DML command is a call to a FORTRAN subroutine. Thus the DML can be considered as an extension to the FORTRAN language. This extended FORTRAN provides for all varieties of data structures and furnishes powerful data handling capacities.

In order to demonstrate DML commands and how they may be used, a detailed discussion is provided of a FORTRAN subroutine which executes a pivot operation. This subroutine assumes that a matrix has been stored in a data base whose logical structure is shown in Figure 6a. Figure 6b gives the DDL of this structure. We must point out that no storage is allocated for any zero valued coefficient before, during or after the pivot operation. Figure 7 displays the pivot subroutine with input arguments for the variable leaving the basis (BVAR) and the variable entering the basis (NBVAR).



a.

RECORD	BASE			
ITEM	NAMB	CHAR	4	NAME OF BASIC VARIABLE
RECORD	NONB			
ITEM	NAMN	CHAR	4	NAME OF NON-BASIC VARIABLE
RECORD	COEF			
ITEM	VAL	REAL	1	COEFFICIENT VALUE
SET	SB	SORTED	NAMB	
OWNER	SYST			
MEMBER	BASE			
SET	SN	SORTED	NAMN	
OWNER	SYST			
MEMBER	NONB			
SET	ROW			
OWNER	BASE			
MEMBER	COEF			
SET	COL			
OWNER	NONB			
MEMBER	COEF			
SET	SC			
OWNER	SYST			
MEMBER	COEF			

b.

Figure 6. Schema and DDL Used by Algorithm of Figure 7.

Hollerith arguments of DML commands refer to data item types, record types or sets that are defined in the schema; all other arguments are FORTRAN variables. The FMSK command (Find Member based on Sort Key) locates the member of the set SB whose sort key (the data item type NAMB) has the value given in the FORTRAN variable BVAR. Each record occurrence in the data base is identified by its own unique data base key. GKM (Get Key from Member) gets the key of the member of the set SB (i.e., the member which has just been located by FMSK) and places that value in the FORTRAN variable KEYB. Next, SFM (Set Field based on Member) sets the value of NAMB in the member occurrence of SB to the value of the FORTRAN variable NBVAR. In these three commands, we have removed BVAR from the set of basic variables, and we have entered NBVAR into the basis. Similarly the next three commands remove NBVAR from the set of non-basic variables and replaces it with BVAR; KEYB now contains the key of the formerly basic variable and KEYN is the key of the formerly non-basic variable.

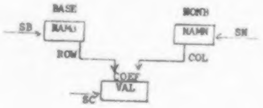
Now we find and update the pivot element. The command SOM (Set Owner of one set based on the Member of another set) sets the owner of the set ROW, based on the current member of the set SB. Recall that the current member of the set SB is the formerly basic variable. As a result of the SOM, the current member of ROW is an occurrence of the record type COEF. The next two commands are used to determine if this coefficient was a member of the COL (column) owned by the formerly non-basic variable. SMM sets the member occurrence of COL based on the current member of ROW. Since a set is a strictly one-to-many relationship, having a member of COL immediately gives us its unique owner. GKO gets the key of this owner and returns it in the variable KEYC. We check KEYC against KEYN to find if we have located the coefficient which was the member of the COL owned by the previously non-basic variable. If this is not the case, then we call FNM (Find Next Member) which locates the next occurrence of coefficient that was in the ROW of the formerly basic variable. If there is no next member then an error has occurred. If a next coefficient of the formerly basic row is found, then as before, we determine with which column that coefficient is associated. Upon finding the pivot element, we update it appropriately and retrieve its key (i.e., the address of its location in the data base).

In order to update coefficients in the pivot column, we first set the owner of COL based on KEYN by using the SOK command. Usage of the other commands appearing in this procedure has already been illustrated with the exception of DRM (Delete Record based on Member). As can be seen from the code, if the updated value (Y) of a coefficient is zero then DRM is used. This command deletes the current member of COL (i.e., the occurrence of COEF whose updated value would have been zero).

Upon udating an element in the pivot column we proceed to update each coefficient in the column element's row. The commands for this are much the same as those given above. There are two new points here. FFM finds the first member of a specified set; if FNM returns a negative value in the IERR variable then there are no more members in the set being considered. As the code indicates, for each non-zero coefficient in the pivot row, we update the corresponding coefficient in the pivot column element's row. Now if there is no such corresponding coefficient, (i.e., the coefficient was zero) then one must be created and given the proper value. The command CR creates a record occurrence of the type specified in its first argument. So statement 60 allocates storage for a record occurrence of the type COEF; KEYNB is returned from CR containing the unique data base key for the created record occurrence. The next command, SFR (Set Field of Record), inserts the value of X*Y into the VAL field (i.e., data item type) of the just created record occurrence of COEF. Finally this record occurrence must be associated with the proper row and column through the sets ROW and COL. This is accomplished by AMS (Add Member to Set). For instance, the final AMS shown adds the record occurrence of COEF to the appropriate owner occurrence of the set COL.

If there is no need to create an occurrence of COEF, then the existing tableau element is updated in a straightforward manner. If the updated value is zero then the record occurrence of COEF is deleted; the space which it has occupied is returned to a pool of available space for future uses of the CR command. Finally after coefficients of the row associated with each pivot column element (except for the pivot element itself) are updated, the pivot row coefficients are properly adjusted.

The purpose of the foregoing example is to be be suggestive of the nature and capacities of a DML. A comprehensive description of the GPLAN DML, including physical implementation and operation, is given in [2]. Observe that the DML obviates any need for overlays or paging with respect to data storage. In the preceding example there was no need to use an array. In addition the programmer is not required to have explicit knowledge of any physical pointers. In the pivot example, the programmer never sees a physical pointer; all data manipulation is dealt with on a logical level using the logical structure of the schema. Each DML com-



```
      SUBROUTINE PIVOT (BVAR,NBVAR)
      INTEGER BVAR
      DATA EPSLON/1.E-8/
C
C     TAKE BVAR OUT OF BASIS, REPLACE WITH NBVAR
C
        CALL FMSK (2HSB,  BVAR, IERR)
        CALL GKM  (2HSB,  KEYB, IERR)
        CALL SFM  (4HNAMB, 2HSB, NBVAR, IERR)
        CALL FMSK (2HSN,  NBVAR, IERR)
        CALL GKM  (2HSN,  KEYN, IERR)
        CALL SFM  (4HNAMN, 2HSN, BVAR, IERR)
C
C    FIND AND UPDATE PIVOT ELEMENT
C
        CALL SOM  (3HROW, 2HSB, IERR)
   10   CALL SMM  (3HCOL, 3HROW, IERR)
        CALL GKO  (3HCOL, KEYC, IERR)
        IF (KEYC. EQ. KEYN) GO TO 20
        CALL FNM (3HROW, IERR)
        IF (IERR) 999,10,999
   20   CALL GFM (3HVAL, 3HROW, PIV, IERR)
        CALL SFM (3HVAL, 3HROW, 1./PIV, IERR)
        CALL GKM (3HROW, KEYP, IERR)
C
C   ITERATE THRU, UPDATE PIVOT COLUMN, DIVIDE EACH COEFFICIENT IN PIVOT COLUMN BY PIVOT ELEMENT
C
        CALL SOK (3HCOL, KEYN, IERR)
   30   CALL GKM (3HCOL, KEYC, IERR)
        IF (KEYC.EQ KEYP) GO TO  105
        CALL GFM (3HVAL, 3HCOL, Y, IERR)
        Y = -Y/PIV
       IF (ABS(Y).GT.EPSLON) GO TO 35
       CALL DRM (3HCOL, IERR)
       Y = 0.0
       GO TO 30
   35   CALL SFM (3HVAL, 3HCOL, Y, IERR)
C
C   DETERMINE KEY OF THIS COLUMN ELEMENT'S ROW (CER)
   36   CALL SMM (3HROW, 3HCOL, IERR)
        CALL GKO (3HROW, KEYR, IERR)
```

Figure 7. Pivot with DML

187

```
C
C     NOW, ITERATE THRU THE PIVOT ROW
C         CALL SOK (3HROW, KEYB, IERR)
      40  CALL SMM (3HCOL, 3HROW, IERR)
          CALL GKM (3HCOL, KEYFR, IERR)
          IF (KEYFR. EQ. KEYP) GO TO 95
          CALL GFM (3HVAL, 3HCOL, X, IERR)
C
C     DOES COL HAVE AN ENTRY IN THIS COLUMN
C         CALL FFM (3HCOL, IERR)
      50  CALL SMM (3HROW, 4HCOL, IERR)
          CALL GKO (3HROW, KEY, IERR)
          IF (KEY.EQ. KEYR) GO TO 70
          CALL FNM (3HCOL, IERR)
          IF (IERR) 60, 50, 999
C
C     NO CORRESPONDING ENTRY - CREATE ONE
C     60  CALL CR (4HCOEF, KEYDB, IERR)
          CALL SFM (3HVAL, 4HCOEF, X*Y, IERR)
          CALL AMS (2HSC, 4HCOEF, IERR)
          CALL SOK (3HROW, KEYR, IERR)
          CALL AMS (3HROW, 4HCOEF, IERR)
          CALL AMS (3HCOL, 4HCOEF, IERR)
          GO TO 90
C
C     UPDATE TABLEAU ELEMENT
C     70  CALL GFM (3HVAL, 3HCOL, W, IERR)
          W = W + X*Y
          IF (ABS (W). LT. EPSLON) GO TO 80
          CALL SFM (3HVAL, 3HCOL, W, IERR)
          GO TO 90
C
C     DELETE ZERO OCCURRENCE
C         80 CALL DRM (3HCOL, IERR)

C     GO ON TO NEXT ELEMENT IN PIVOT ROW
      90  CALL SMK (3HROW, KEYFR, IERR)
      95  CALL FNM (3HROW, IERR)
          IF (IERR) 100, 40, 999
C
C    FIND NEXT VALUE IN PIVOT COLUMN
C    100  CALL SMK (3HCOL, KEYC, IERR)
     105  CALL FNM (3HCOL, IERR)
     106  IF (IERR) 110, 30, 999

C
C    FINALLY, UPDATE PIVOT ROW ITSELF
     110  CALL SOK (3HROW, KEYB, IERR)
     120  CALL GKM (3HROW, KEYC, IERR)
          IF (KEYC. EQ. KEYP) GO TO 130
          CALL GFM (3HVAL, 3HROW, X, IERR)
          CALL SFM (3HVAL, 3HROW, X/PIV, IERR)
     130  CALL FNM (3HROW, IERR)
          IF (IERR) 140, 120, 999
C
C
C    ERRORS
     999 CALL ERR (IERR, 3, 0)
         CALL ABORT


     140 RETURN
         END
```

Figure 7. (continuation) .

mand is a statement in terms of the logical structure and it automatically executes the cumbersome physical details implied in that statement.

PRACTICAL CONSIDERATION

It is not suggested that the foregoing subroutine be used in a practical sense. Since the tone of this paper is basically tutorial, we have chosen the familiar pivot operation as a vehicle for conveying the fundamental ideas and methods of data structuring and data manipulation. The example serves to illustrate all of the DML commands which are typically used in process of data base management. A major difficulty with the simplex method is that an originally sparse matrix is usually altered such that this sparseness vanishes. We thereby lose the benefits a data base management system can provide in terms of storing and manipulating sparse matricies.

From a practical standpoint the revised simplex method furnishes certain well-known computational advantages [7]. The initial problem description is not subject to modification; rather than explicitly pivoting, we successively update the inverse of each new basis. The current basis at any step may be computed by taking the product of elementary matricies, were each elementary matrix is the identity except for one column. Thus a data structure capable of supporting the revised simplex method must be able to store both the original problem matrix and the non-trivial columns of elementary matricies which compose the product form of the basis inverse.

The structure of Figure 4 suffices for the original problem. Recall that no storage is allocated for zero valued coefficients, so substantial storage savings are realized as problems tend toward sparseness. There is also a savings in processing; if a coefficient is not stored then it cannot be considered for processing. These savings result regardless of whether there exists a special pattern of sparseness (e.g. band matricies) or whether there is no discernable pattern. That is, the savings depend only upon the percentage of zero elements and not upon their arrangement.

There are several ways to store columns of the elementary matricies. In situations where it is undesirable or not possible to store these in core, a data structure similar to that of Figure 8 could be used. As the structure indicates, an occurrence of PROBLEM has many occurrences of EXPRESSION, of VARIABLE, and of INVERSE associated with it. The former two describe the initial matrix, which is not subject to change. At the beginning there are no occurrences of INVERSE; however one occurrence is created and added to the set PRODUCT at each iteration of the revised simplex algorithm. In the DDL the data item type COLUMN-VALUE is defined to be a vector. So a DML reference to an occurrence of this item will involve an entire vector of column values. For instance the command GFM (4HCOLV, 4HPROD, DATA, IERR) will fill the array DATA with the data vector of values from COLV for the current member of the set PROD. The position of a given column in its elementary matrix is stored as the value of COLUMN-NO. Finally we must be able to access the columns in the order in which their respective elementary matricies were generated. Since we can declare (in the DDL) an ordering for the members of a set, we specify PRODUCT to be ordered on a FIFO basis (First-In-First-Out). Thus access to occurrences of INVERSE for a particular occurrence of PROBLEM is based upon the order in which those occurrences were originally added to the PRODUCT set. It should be clear that the DML commands used for implementation of the revised simplex method in a data base context are the same as those presented in the pivot example (except for DRM, which is not needed for the revised simplex method).

We realize of course that the revised simplex and the product form are well-known, and that numerous specialized data packing schemes have been developed for treating sparse matricies. What is interesting is the independent development of data handling mechanisms which are, in a sense, general packing schemes. The term 'general' refers to the ability of a single data base management system (e.g., GPLAN) to support data (numeric and non-numeric) storage and analysis for a broad variety application areas. Past and present projects involving the GPLAN data management system include the areas of water quality control and planning, forestry, health care, material requirements planning, PERT management, and socio-economic research. Each area requires data structures that embody that area's relevant characteristics. Typically, each area also requires that several types of analysis be performable. In the water quality area for instance, analyses of the following types have been found to be useful: simulations, linear programming, non-linear programming, and various statistical analyses [8]. Programs to perform any of these may be implemented with the DML.

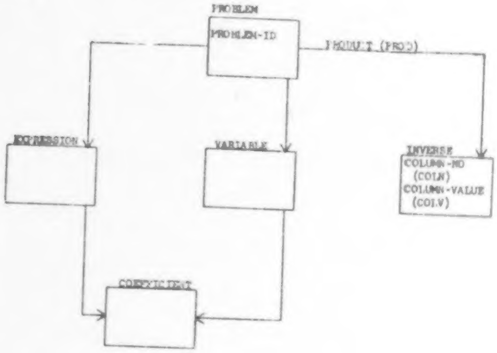


Figure 8. Possible Data Structure for the Revised Simplex Method with Product Form of the Basis Inverse.

We have indicated in the foregoing discussion how data base management techniques may be used by programmers in LP development, in such a way that efficiencies of special data packing schemes are realized. Similarly this same data manipulation language may be used to accomplish any other programmable analysis. Thus data base management tools allow a single data base, containing both numeric and non-numeric data, to support requests of various users for a variety of analyses and reports. An LP package is insufficient for this sort of task, although it may be quite efficient in what it does do. On the other hand, an installation which possesses a data management system, can devise its own LP routines without regard to special packing schemes and buffering procedures for interface with auxiliary memory. These are handled automatically by the data management system, and its general packing scheme provides substantial savings in the event of sparseness. Another important feature of the data base management system is its capacity to support a query system that allows non-programmers to utilize a data base and analytic routines.

QUERY LANGUAGE

Whereas GPLAN/DML requires that a user write programs in a host language with the utilization of pertinent DML commands, the GPLAN Query System does not require one to be a programmer in order to utilize the data base for purposes of display or execution of large application routines. The user needs merely to specify what is to be done; there is no statement of the procedures to be followed in order to accomplish the task. Examples of very simple commands are

```
LIST COEFFICIENT.SOURCE FOR VARIABLE.ID = 'X1' and
CONSTRAINT.ID = 'R3'

LIST VARIABLE.ID AND CONSTRAINT.ID FOR COEFFICIENT.
SOURCE = 'TEST 1'
```

Upon receipt of such commands, the query system analyzes the request, sets up the necessary DML commands, executes those commands and supplies the requested data values. The system is designed such that it permits the selective (or unconditional) retrieval of any data configuration. Moreover, it permits execution of application routines using any desired (and germane) data from the data base. The fundamental query syntax is

```
<COMMAND> <RETRIEVAL CLAUSE> <CONDITIONAL CLAUSE>
```

The command indicates which application routine is to be executed; in the queries above, LIST indicates that a report generator is to be executed. In the retrieval clause the user specifies what data are to be used for execution; this retrieval is dependent on conditions specified in the conditional clause.

A user of the query language is allowed to present arbitrarily complex retrieval clauses. Not only may this clause contain the names of data items to be retrieved, but arithmetic operations (using literals or data items) and both single and multivariate functions may also be introduced. The conditional clause is composed of a Boolean expression which may contain data item names, literals, arithmetic operators, relational operators, logical operators, single-variable functions and multivariate functions. The query language also permits the use of noise words, synonyms and various other cosmetic features for the convenience of the user.

A conceptual overview of the standard GPLAN system is portrayed in Figure 9. The library of application routines is composed of two sections: standard routines and special routines. The standard library of applications consists of routines to generate reports and plots and to perform linear regressions, statistical analyses, and data modification. The library of special applications may include such routines as special report generators, linear and non-linear optimization programs. The issue of interfacing such optimization routines with a network data base is discussed in [4]. One method of interface has been demonstrated in the pivot example.

DATA HANDLING FOR MATHEMATICAL PROGRAMMING

In this section we examine the implications of data base management techniques for those who implement mathematical programming algorithms and for those who make use of such implementations. In so doing, we utilize the distinctive GPLAN features

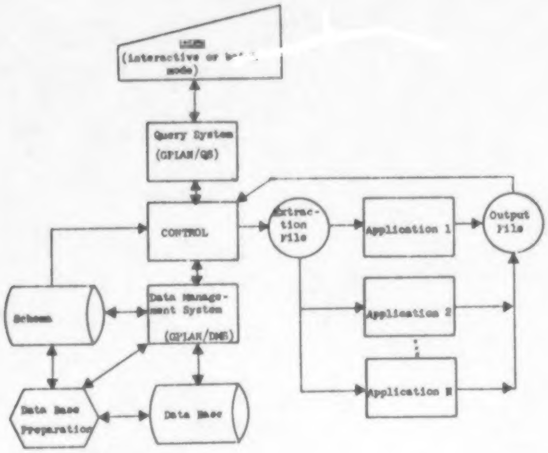

(interactive or batch
mode)
Query System
(GPLAN/QS)
CONTROL
Extraction
File
Application 1
Output
File
Schema
Data Management System
(GPLAN/DMS)
Application 2
Application N
Data Base
Preparation
Data Base

Figure 9. GPLAN System

of the network data base structure, the language for non-programmer interface with a data base (DML) and the query language that allows non-programmers to effectively use a data base and pertinent application routines. Not only does the GPLAN Framework allow for the obvious, i.e., the solution of linear and non-linear optimization problems; it also addresses the following considerations, which may perhaps be more subtle, but are certainly of practical significance [5].

1. The resolution of erroneous formulations.
2. Treatment of coefficients which are themselves functions.
3. Situations wherein matrices contain common data.
4. Storage of sparse matrices.
5. Utilization of data to produce timely, non-routine reports other than the report furnished by a general mathematical programming routine.
6. Ability of a data base to support other varieties of application routines (e.g., simulations, regressions, etc.) in addition to mathematical programming.

The modus operandi for effective accomodation of each of these attributes may be found in [4]. Only a very brief treatment can be given here. With respect to the first point, when erroneous coefficients or improper formulation is suspected the ready availability of information with regard to coefficient sources and currency is important. This has obvious implications for the way in which data is organized and retrieved. In this connection the query system provides a convenient tool for data editing.

Concerning the second attribute, it is not uncommon that coefficients are the results of functions that have been evaluated on the basis of some other data. There may even be alternate functional forms (or alternate data sets for evaluating a function) which suggests the need for a facile method of interfacing desired functions with the math programming routines which depend upon them. Indeed a linear programming routine may be viewed as a function that requires (recurring) evaluation in order to execute a non-linear programming routine [6]. One technique for handling this situation involves an elimination of the distinction between data and function, with respect to data base construction; i.e., functions are treated as data and included in the data base structure.

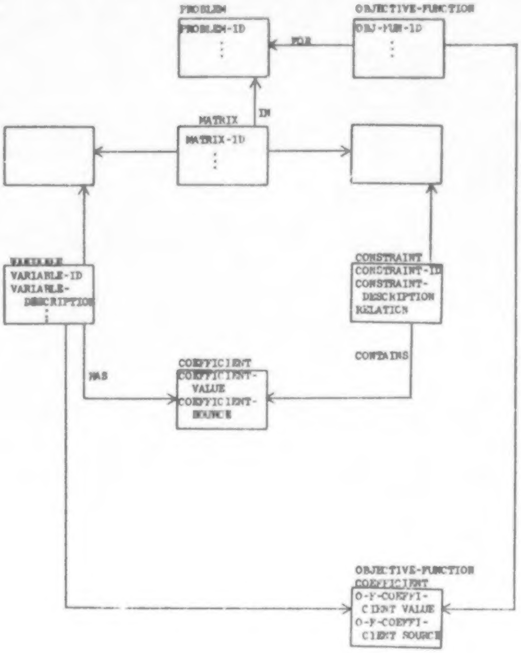

PROBLEM
PROBLEM-ID
OBJECTIVE-FUNCTION
OBJ-FUN-ID
FOR
IN
MATRIX
MATRIX-ID
VARIABLE
VARIABLE-ID
VARIABLE-
DESCRIPTION
CONSTRAINT
CONSTRAINT-ID
CONSTRAINT-
DESCRIPTION
RELATION
CONTAINS
HAS
COEFFICIENT
COEFFICIENT-
VALUE
COEFFICIENT-
SOURCE
OBJECTIVE-FUNCTION
COEFFICIENT
O-F-COEFFI-
CIENT VALUE
O-F-COEFFI-
CIENT SOURCE

Figure 10. Extended Logical Data Structure

Attribute number three is important for cases where there is interest in several matrices, which are not entirely distinct with respect to constraints and variables. For example, we may be investigating a problem which has multiple plausible formulations, some pairs of which share constraints (and therefore variables). Care must be taken to assure that updates to constraints in one matrix are reflected in other matrices which share these constraints; this is not a trivial matter where large volumes of data are involved. Data structure that can support this situation is given in Figure 10. This structure allows us to store a constraint only once, and at the same time indicate that it is to be included in an arbitrary number or matrices. This avoidance of redundancy averts the potential for inconsistencies. A detailed description of this structure at the occurrence level may be found [4]. Figure 11 displays verbatim examples of the kinds of queries which could be submitted, if the data structure is as shown in Figure 10. The first three queries show how queries may be phrased to locate and rectify errors in the data. The next two queries demonstrate ways in which are LP problems may be formulated for execution. The final query asks for basic statistics on the coefficient values of variable 497 in matrix 7.

```
LIST CONSTRAINT.ID, COEFFICIENT.VALUE AND COEFFICIENT.SOURCE FOR VARIABLE.ID = 1347,
COEFFICIENT.VALUE < 1.0 OR COEFFICIENT.VALUE > 9.327, AND MATRIX.ID = 4

CHANGE COEFFICIENT.VALUE TO 8.27 WHEN VARIABLE.ID = 1347, CONSTRAINT.ID = 39,
AND MATRIX.ID = 4

CHANGE COEFFICIENT.VALUE TO LOG (COEFFICIENT.VALUE)*1.14 IF VARIABLE.ID = 58
AND MATRIX.ID = 21

RUN LP FOR MATRIX.ID = 17 AND OBJ-FUN-ID = 5

RUN LP FOR PROBLEM.ID = 38

STAT COEFFICIENT.VALUE FOR VARIABLE.ID = 497 AND MATRIX.ID = 7
```

Figure 11. Sample Queries

Many real-world applications entail the utilization of sparse matrices. In the effort to avoid storing zeros, many schemes have been devised for packing (and unpacking) non-zero coefficients into arrays. As already shown in the pivot example, the GPLAN concept allows all matrices (sparse or otherwise) to be accomodated by a single simple logical structure, which realizes substantial storage savings if a matrix happens to be sparse. Only non-zero coefficients need to be stored; and storage space is neither used nor even allocated for zero coefficients.

Presumably managerial decisions are not based soley upon the output of mathematical programming routines. The fifth consideration indicates the need for a facility to generate other reports from a data base and frequently these are non-standard in terms of the types and configurations of data that are retrieved. The GPLAN query system allows the selective retrieval of any configuration of data as a result of typing an English-like, non-procedural query at a computer terminal. This obviates the necessity of writing a program every time a new type of report is needed. This concept can be extended to include not only retrieval, but also the execution of large application routines that are not of the mathematical programming variety. Furthermore, such executions can be accomplished through the query language. The result is a situation wherein a network data base can support a broad spectrum of analyses for both programmers and non-programming users.

CONCLUSION

The major intent has been to introduce fundamental concepts from the realm of data base management and to suggest their potential for contribution to the solution of mathematical programming problems. This contribution is considered with respect to both the implementors and users of mathematical programming algorithms. GPLAN, a generalized data base management and query system, was described. It is essential to observe that there is invariably a tradeoff between specialized and general systems. The former are generally more efficient due to their limited flexibility; the latter are typically not as efficient due to their great flexibility. Both inefficiency and inflexibility have costs. GPLAN provides a single mechanism which may be used to treat the spectrum of special cases and to support a variety of applications and users working with the same data base. As an exercise, this general system could be tailored to meet only special needs, in circumstances where flexibility is unimportant and efficiency is paramount.

Historically, the trend has been for each math programming implementor to, in effect, devise data management routines. We submit that advances (both past and future) in the data base management field can provide valuable concepts, techniques and tools for the data handling aspects of mathematical programming.

REFERENCES

1. CODASYL: Data Base Task Group Report, ACM, April, 1971.
2. Haseman, W.D. and Whinston, A.B., Introduction to Data Management, Richard D. Irwin, Homewood, IL, 1977.
3. Bonczek, R.H., Holsapple, C.W., and Whinston, A.B., "Extensions and Corrections for the CODASYL Approach to Data Base Management," International Journal of Information Systems (forthcoming), 1976.
4. Bonczek, R.H., Holsapple, C.W., and Whinston, A.B., "Mathematical Programming Within the Context of a Generalized Data Base Management System," Krannert Institute Paper No. 578, Purdue University, November 1976.
5. White, W.W., "A Status Report on Computing Algorithms for Mathematical Programming," ACM Computing Surveys, September 1973.
6. Graves, G., Pingry, D. and Whinston, A., "Water Quality Control: Non-linear Programming Algorithm," Revue Francaise d'Automatique, Informatique et Recherche Operationnelle, Oct., 1972
7. Dantzig, G.B., Linear Programming and Extensions, Princeton University Press, 1968.
8. Holsapple, C.W., and Whinston, A.B., "A Decision Support System for Area-Wide Water Quality Planning," Socio-Economic Planning Sciences, forthcoming.

COVERGENCE OF THE DIAGONALIZED METHOD OF MULTIPLIERS

R. H. Byrd
DEPARTMENT OF MATHEMATICAL SCIENCES
The Johns Hopkins University

Abstract. Recently, modifications of the method of multipliers have been proposed which do not require exact minimization in the intermediate unconstrained problem. It can be shown that any Lagrange multiplier updating scheme which yields quadratic convergence with exact minimization retains this rate if only two steps of a Newton iteration are taken. It can also be shown that a quasi-Newton iteration in conjunction with the multiplier update proposed by Tapia yields a superlinearly convergent algorithm. If the penalty constant is allower to increase to infinity at a certain rate, simpler multiplier update formulas will exhibit good convergence rates.

Recently there has been much research in solution of constrained optimization problems by the method of multipliers. This method was first proposed by Hestenes [1] and by Powell [2] independently in 1969. It involves solving the problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{such that} & g(x) = 0 \end{array} \qquad (1)$$

where $f: R^n \to R^1$ and $g: r^n \to R^m$ by successively minimizing the augmented Lagrangian

$$L(x,\lambda,c)=f(x)+\lambda^T g(x)+\frac{1}{2}cg(x)^T g(x)$$

in x. If x^* is the solution of problem (1) then there exists a unique $\lambda^* \in R^*$ such that $\nabla_x L(x,^*,\lambda^*,c) = 0$, where ∇_x we mean the gradient taken with respect to the variable x. We will assume throughout that $m < n$, that $\nabla g(x^*)$ is of full rank and that $\nabla_x^2 L(x^*,\lambda^*,c)$ is invertible. It can be shown, Buys [3] that there exists a constant c^o such that for $c > c^o$, x^* is an unconstrained local minimum of $L(x^*,\lambda^*,c)$ and for λ in a neighborhood of λ^*, $L(x,\lambda,c)$ has a local minimum in x.

The method of multipliers procedes as follows:

Step Choose an initial approximation λ^o to λ^*, and an initial c^o.

Step 2: Choose x_k^k to be the minimizer of $L(x,\lambda^k, c^k)$.

Step 3: Let $c^{k+1} = P(x^k,\lambda^k,c^k)$
$\lambda^{k+1} = U(x^k,\lambda^k,c^{k+1})$

Step 4: Return to step 3 with $k = k+1$.

The function U may be referred to as a multiplier update formula. The function P is sometimes chosen simply so that c^k is large enough that a minimizer of L exists, while for other schemes c^k tends to infinity near the solution.

A number of multiplier update formulas have been proposed. We consider the following:

$$U_{HP}(x,\lambda,c) = \lambda+cg(x) \qquad (2)$$

$$U_B(x,\lambda,c) = \lambda+[\nabla g(x)^T\nabla_x^2 L(x,\lambda,c)\nabla g(x)]^{-1}g(x) \qquad (3)$$

$$U_P(x,\lambda,c) = -[\nabla g(x)^T\nabla g(x)]^{-1}\nabla g(x)^T\nabla f(x) \qquad (4)$$

$$U_T(x,\lambda,c) = [\nabla g(x)^T\nabla_x^2 L(x,\lambda,c)^{-1}\nabla g(x)]^{-1} [g(x) - \nabla g(x)^T\nabla_x^2 L(x,\lambda,c)^{-1}\nabla f(x)] - cg(k) \qquad (5)$$

Formula (2) is the one originally proposed by Hestenes and by Powell. Rupp [4], and Bertsekas [5] have shown that for c^k sufficiently large the Hestenes-Powell update gives arbitrarily good linear convergence. Formula (3) was proposed by Buys [3] who showed that, if x^k is considered as a function of λ, (3) is equivalent to one Newton iteration on the problem $g(x(\lambda)) = 0$, and this gives quadratic convergence. Formula (4) is just the projection of ∇f onto the linear span of $\nabla g_1(x),\ldots,\nabla g_m(x)$. Its use has been proposed by Haarhoff and Buys [6], Miele et at [7], and Fletcher [8]. Formula (5) was originally proposed by Tapia [9] as a consequence of a general theory for solving constrained problems.

Finding x^k which minimizes $L(x,\lambda^k,c^k)$ exactly is of course impossible and even finding a good approximation may be very costly. It is natural to consider algorithms involving simple approximatations to the minimizer.

Bertsekas [5] considers an implementation with the Hestenes-Powell update where the minimization is carried out until $\|\nabla_x L\|$ is less than some tolerance, and shows that for the tolerance correctly chosen one may obtain convergence as good as in the case of exact minimization although nothing may be said about how difficult that tolerance is to achieve for large penalty constants.

Another strategy is to take a small number of iterations of a minimization algorithm, a strategy referred to by Tapia as a diagonalized multiplier method [9]. Such a strategy has also been proposed by Haarhoff and Buys [7] and implemented by Miele et at [6].

We consider first the case of algorithms implemented with bounded penalty constants i.e. $c^k \leq \bar{c}$ for all k. If a multiplier update formula such as that of Buys is used one needs second order information about the functions involved so it is reasonable to consider methods in the unconstrained minimization phase which generate and use the second derivative or an approximation to it.

We will refer to an algorithm where the point x^{k+1} is determined by j steps of some iterative algorithm as a j step diagonalization.

Based on different considerations Tapia [9,10,11] proposed an algorithm which is equivalent to a one-step diagonalization using Newton's method. That is, the point x^k is determined by

$$x^k = x^{k-1} - \left[\nabla_x^2 L(x^{k-1},\lambda^{k-1},c^k)\right]^{-1}\nabla_x L(x^{k-1},\lambda^k,c^k)$$

and λ^k is chosen according to Tapia's update: $\lambda^k = U(x^{k-1},\hat{\lambda},c^k)$ where $\hat{\lambda} = U_p(x^{k-1})$. Tapia shows this method has the following convergence properties (11).

Proposition 1. If the one-step diagonalized method of multipliers is implemented with Tapia's update and Newton's method, the sequence $\{x^k\}$ is uniformly quadratically convergent to the solution x^*.

This result is obtained by noting that Tapia's algorithm is essentially equivalent to using Newton's method on the problem $\nabla_x L(x,\lambda,c) = 0$, $g(x) = 0$.

By uniform quadratic convergence of an algorithm we mean there exist constants $M, \nu > 0$ such that if $\|x^0 - x^*\| < \nu$, then $\|x^{k+1} - x^*\| \leq M\|x^k - x^*\|^2$ for all k.

The notion of uniform quadratic convergence is slightly stronger than Q-quadratic convergence, but almost all iterations which are Q-quadratically convergent are uniformly quadratically convergent Using the projection formula for $\hat{\lambda}$ allows us to get uniform quadratic convergence. If we let $\hat{\lambda} = \lambda^{k-1}$ we only get R-quadratic convergence of $\{x^k\}$.

Since both the Buys multiplier update formula and the Tapia formula give quadratic convergence when x^k is obtained by exact minimization it is natural to ask how the Buys update behaves under diagonalization. If we consider the question of quadratic convergence in the space of both variables, x and λ we get the following result.

Proposition 2. Consider the method of multipliers implemented with a given multiplier update formula and with bounded penalty constants. Uniform quadratic convergence in (x,λ) occurs with exact minimization if and only if it occurs with a two-step diagonalization using Newton's method.

Proof. See [12].

Since both the Tapia update and the Buys update give quadratic convergence for exact minimization, it follows then that one may get the same convergence for a two-step diagonalization with Newton's method, and for such update it seems nothing is contributed to convergence rate by further iterations.

Since using Newton's method required knowledge of the second derivatives of the functions involved and requires a large number of function evaluations at each step it would be advantageous if a quasi-Newton method could be used in place of Newton's method in the determination of x^k. If this were done with one of the quadratically convergent updates such as that of Tapia or Buys it would be necessary to use an approximation to $\nabla_x^2 L(x,\lambda,c)$ in the multiplier update formula. Consider then the following iteration:

$$\begin{aligned}
\lambda^{k+1} &= \left[\nabla g(x^k) B_k^{-1} \nabla g(x^k)\right]^{-1}\left[g(x^k) - \nabla g(x^k)^T B_k \nabla f(x^k)\right] - c_k g(x^k)\\
x^{k+1} &= x^k - B_k^{-1}\nabla_x L(x^k,\lambda^{k+1},c^k)\\
B_{k+1} &= V\big(x^{k+1} - x^k, \nabla_x L(x^{k+1},\lambda^{k+1},c^k) - \nabla_x L(x^k,\lambda^{k+1},c^k), B_k\big)\\
c_{k+1} &= P(x^{k+1},\lambda^{k+1},c^k)
\end{aligned} \tag{6}$$

The function $V(s,y,B)$ is the Hessian update function for some quasi Newton method. The analysis of such an algorithm is somewhat different for each possible Hessian update function but Tapia has proved the following result for standard quasi-Newton methods [11].

Proposition 3. Consider the method of multipliers implemented as in (6) with bounded penalty constants and with the function V being any one of the Broyden, Davidon-Fletcher-Powell, Broyden-Fletcher-Goldfarb-Shanno or Powell Symmetric Broyden, Hessian update functions. The sequence of points (x^k,λ^k) will then be locally and superlinearly convergent to (x^*,λ^*).

This result shows that the algorithm ourlined in (6) is a relatively efficient strategy. It is possible that there would be some advantage in taking more than one quasi-Newton step in determining x^k, however by this result the convergence rate would still be superlinear.

It seems likely that if a quasi-Newton method were used with the Buys update in some diagonalized scheme that superlinear convergence would result. However, it has not been possible to prove this yet. It is interesting to note that according to the above result, the matrix B_k works well as an approximation to the Hessian in the multiplier update formula even though it is known that B_k does not converge to the Hessian in general.

Up to this point we have been concerned with algorithms where the penalty constant is bounded throughout the iteration. However, it is well known that in some cases the rate of convergence is improved as the penalty constant goes to infinity. As mentioned prefiously Bertsekas [5] shows arbitrarily first linear convergence for exact and approximate mimimization in these cases. Miele et al [6] have done numerical experiments involving the use of the projection update (equation 5) in various diagonalized schemes using Newton's methods and quasi-Newton methods. By increasing the penalty constant at the appropriate rate they get convergence which appears R-superlinear.

In addition to the projection update we consider multiplier update formulas $U(x,\lambda,c)$ with the property

$$\frac{\partial}{\partial\lambda} U(x^*,\lambda^*,c) = 0.$$

We will refer to such formulas as local multiplier approximation formulas. Both the projection update and the Tapia update fall into this class. Note that it is essential that such a formula have the property $U(x^*,\lambda^*,c) = \lambda^*$.

In the case of exact minimization, as the penalty constant c goes to infinity the various updates mentioned tend to merge together and all give good convergence. In the case of local approximation formulas, the author shows in [13] that if c^k is chosen to be $\gamma \| g(x^{k-1}) \|^{-\alpha}$ where $\gamma > 0$ and $0 < \alpha \leq 1$ the iteration is locally convergent with Q-order $1 + \alpha$.

However if such a scheme for choosing penalty constants is used in a one-step diagonalization with Newton's method, convergence becomes slower when α is close to one. If $\alpha = 1$ the method is very slow and may not converge at all. The problem is that when the penalty constant increases too fast the Newton iteration becomes unstable. This slow convergence occurs even if the normally quadratically convergent Tapia update is used.

These difficulties may be avoided in a way leading to efficient algorithms in two ways. If a two step diagonalization is used one gets again the same convergence rate as with exact minimization. At best with $\alpha = 1$ the method is quadratically convergent. Alternatively a one step diagonalization may be used with $\alpha = \frac{1}{2}$ in this case the order of convergence is $1\frac{1}{2}$. This can be shown to be the optimum value of α for a one step diagonalization [13].

The above development seems to indicate that a one or two step diagonalization is as effective as exact minimization in ensuring fast convergence. Alternatively if one is using a scheme demanding a certain degree of exactness in the minimizer these results give some indication of how much effort is required to achieve this exactness.

BIBLIOGRAPHY

1. Hestenes, M.R., "Multiplier and gradient methods," Journal of Optimization Theory and Applications, Vol 4, pp. 303-320, 1969.

2. Powell, M.J.D., "A method for nonlinear constraints in minimization problems," in Optimization, Edited by R. Fletcher, Academic Press, London, England, 1969.

3. Buys, J.D., "Dual algorithms for constrained optimization," Ph.D. thesis, Rijksuniversiteit te Leiden, the Netherlands, 1972.

4. Rupp, R.D., "On the combination of the multiplier method of Hestenes and Powell with Newton's method," Journal of Optimization Theory and Applications, Vol. 15, pp. 167-187, 1975.

5. Bertsekas, D.P., "Combined primal dual and penalty methods for constrained minimization," SIAM Journal on Control, Vol. 13, pp. 521-543, 1975.

6. Miele, A., Moseley, P.E., Levy, A.V., and Coggins, G.M., "On the method of multipliers for mathematical programming problems," Journal of Optimization Theory and Applications, Vol. 10, pp. 1-33, 1972.

7. Haarhoff, P.D., and Buys, J.D., "A new method for the optimization of a nonlinear function subject to nonlinear constraints, "Computing Journal, Vol. 13, pp. 178-184, 1970.

8. Fletcher, R., "An ideal penalty function for constrained optimization," Journal of the Institute of Mathematics and its Applications, Vol. 15, pp. 319-342, 1975.

9. Tapia, R.A., "Newton's method for optimization problems with equality constraints," SIAM Journal on Numerical Analysis, Vol. 11, pp. 874-886, 1974.

10. Tapia, R.A., "Newton's method for problems with equality constraints," SIAM Journal on Numerical Analysis, Vol. 11, pp. 174-196, 1974.

11. Tapia, R.A., "Diagonalized Multiplier methods and Quasi-Newton methods for constrained optimization," Department of Mathematical Sciences Technical Report, Rice University, Houston, Texas, June 1975, to appear in Journal of Optimization Theory and Applications.

12. Byrd, R.H. "Local Convergence of the Diagonalized Method of Multipliers." Submitted to Journal of Optimization Theory and Applications.

13. Byrd, R. H., "Local Convergence of the Diagonalized Method of Multipliers," Ph.D. thesis, Rice University, Houston, Texas, 1976.

DIRECT APPROACHES FOR THE MINIMAX PROBLEM

A.R. Conn
Department of Combinatorics and Optimization
University of Waterloo

Introduction.

Consider a system of m real, twice continuously differentiable, and in general, nonlinear, functions

$$f_i(x),\ i \in [M],$$

where $[M] = \{1,2,3...,m\}$ is an index set. Let

$$M_f(x) = \max_{i\in[M]} f_i(x), \qquad (1)$$

$$x = [x_1,x_2,\dots,x_n]^T.$$

The problem under consideration is to locate a point x_0 such that

$$M_f(x_0) \le M_f(x) \qquad : \underline{P}$$

for all points x, at least in a neighbourhood of x_0.

The objective function $M_f(x)$ has discontinuous first partial derivatives at points where two or more of the functions $f_i(x)$ are equal to the maximum. This is true even if the $f_i(x)$ have continuous first partial derivatives.

As an illustration, figure 1 shows the contours for $M_f(x)$ for a purely linear case, viz.

Example 1.

$$f_1 = -10x_1 - 2.5x_2$$
$$f_2 = -10x_1 - 20x_2$$
$$f_3 = -10x_1 + 20x_2 - 40.$$

The points of discontinuity are indicated in figure 1 by dotted lines.

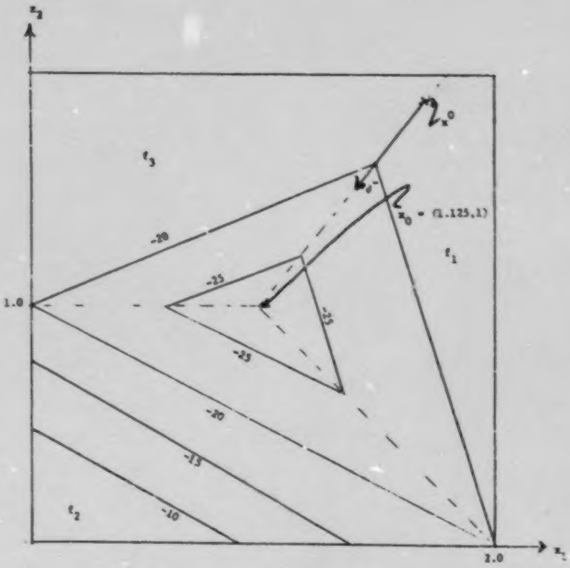


Figure 1. Contours of $M_f(x)$ for Example 1.

We first note that because of the discontinuities in the derivative the well-known gradient type methods cannot be used directly to minimize $M_f(x)$.

Secondly, we note that an equivalent formulation of the problem $\underline{P}$ is (see for example [17])

$$\left.\begin{array}{c}\text{Minimize } z\\ \text{such that, } z - f_i(x) \ge 0,\quad i \in [M],\end{array}\right\}\ \underline{P^*}$$

where z is a new independent variable.

But in general, $\underline{P^*}$ is just a "standard" nonlinear programming problem in n+1 variables, and can be solved accordingly. In fact, this is exactly the approach taken by [16].

Thirdly we note, (as is easy to show, on applying the well-known Kuhn-Tucker conditions to $\underline{P^*}$) that a minimax optimum at x_0 can be characterized (see, for example [17]) by

$$\sum_{i\in E(x_0,0)} u_i \nabla f_i(x_0) = 0, \qquad (2a)$$

$$(\nabla f_i(x_0) = [\frac{\partial f(x_0)}{\partial x_1}, \dots, \frac{\partial f(x_0)}{\partial x_n}]^T)$$

$$\sum_{i\in E(x_0,0)} u_i = 1, \qquad (2b)$$

and

$$u_i \ge 0,\quad i \in E(x_0,\ 0), \qquad (2c)$$

where

$$E(x,\varepsilon) = \{i\in[M] \mid M_f(x) - f_i(x) < \varepsilon\}. \qquad (2d)$$

(i.e., $E(x,\varepsilon)$ is the index set of those functions f_i that are within ε of $M_f(x)$ at x).

We shall now formulate informally an approach to the minimax problem based on three properties cited above.

Suppose we are at x^k. Furthermore, suppose that $f_i(x^k)$, $i = 1,\dots,q$ take on the maximum function value $M_f(x^k)$. As in most optimization approaches our aim is to determine a direction, d^k, say, such that M_f, at x^k, decreases in this direction. It then remains to determine how far to step in the chosen direction and the iteration is defined. It is clear that since the f_i, $i \neq 1,\ldots,q$ are less than $M_f(x^k)$, locally at least, we are only concerned with those functions $f_i(x^k)$, $i = 1,\ldots,q$. We shall term these functions the active functions.

Let us suppose we find a direction d^{k^-} such that f_1, say, decreases. If it should happen that in addition, $f_2,\ldots,f_q$ decrease, then we have a descent direction for M_f. However, suppose that although f_1 decreases, f_2 increases. Then, as is obvious, M_f will in fact increase. Thus what we must do is decrease all the active functions simultaneously.

If we now recall the non-linear programming formulation of the problem, we observe that active functions correspond to active constraints and that f_2 increasing, for example, while z is decreasing, amounts to violating a constraint (assuming f_2 was an active function at the start of the iteration and $z = M_f(x^k)$). Thus a sufficient condition that we have a suitable descent direction for M_f is equivalent to ensuring that no active constraints are violated. [That this is not a necessary condition we will see shortly]. This situation is a familiar occurrence in the context of mathematical programming (see, for example [18] and [15]) and one approach to resolving the difficulty is via projections. [See [8] for further references and details].

An elementary exposition on projection matrices now follows: Suppose we have the following basis for E^n:

$$a_1 \ldots a_q \; b_{q+1} \ldots b_n,$$

where

$$b_i^T a_j = 0 \;\forall i,j.$$

Consider the following n×n matrix,

$$P = I - N(N^TN)^{-1}N^T, \tag{3}$$

where $N = [a_1 \ldots a_q]$, (i.e., N is an n×q matrix of rank q) and I^q is the n×n identity matrix.

It is easily seen that $Pa_j = 0$, $Pb_i = b_i \;\forall i,j$. Furthermore, $P^2 = P$ and we note that P takes any vector in $\mathbb{R}^n$ into $Q^\perp$, where $Q^\perp$ denotes the space spanned by

$$b_{q+1},\ldots,b_n.$$

Now suppose we take P with

$$a_i = \nabla\phi_i(z,x) \quad i = 1,\ldots,q$$

where

$$\phi_i(z,x) = z - f_i(x)$$

and $\nabla\phi_i$ is as defined in equation (15) below, [i.e., $\phi_i \geq 0$ are the inequality constraints of P*]. We note that a_i, and thus P, is independent of z. Furthermore, let us define $d \in \mathbb{R}^{n+1}$ by

$$d = -Pe, \tag{5}$$

where

$$e = [1,0,\ldots,0]^T, \quad e \in \mathbb{R}^{n+1}. \tag{6}$$

Now, in the case where the f_i are linear, we have, by Taylor's theorem

$$\phi_i[z+\lambda d^+, x+\lambda d^-] = \phi_i(z,x) + \lambda d^+ - \lambda[d^-]^T\nabla f_i(x), \tag{7}$$

where $d = [d^+ : d^-]$, $d^+, \lambda \in R^1$, $\lambda > 0$.

But, by construction

$$d^+ - [d^-]^T\nabla f_i(x) = d^T\nabla\phi_i = 0 \quad i = 1,\ldots,q.$$

Thus

$$\phi_i[z+\lambda d^+, x+\lambda d^-] = \phi_i[z,x] \quad i = 1,\ldots,q. \tag{8}$$

More generally, for $i = 1,\ldots,q$, we have that, for concave f_i

$$\phi_i[z+\lambda d^+, x+\lambda d^-] \geq \phi_i(z,x) \quad i = 1,\ldots,q. \tag{9}$$

That is, in the case where the active functions are linear or concave, if the search direction is given by (5), no active constraints in the equivalent non-linear programming formulation are violated.

Furthermore, since $e^Td = -||Pe||^2 < 0$, if $Pe \neq 0$ [which we may assume, for the present], d is a descent direction for z. Thus our objective is realized, i.e., we are reducing M_f.

Referring to figure 1, let us suppose we are at the point x^C where $f_1(x) = f_3(x) = M_f(x)$. The corresponding descent direction for M_f, d^- is indicated on the figure. We see that it is an ideal choice. In particular, we see that, locally at least, the indices of the active functions do not change. This result, of course, is an immediate consequence of equation (8).

In practice, not all the f_i's $i = 1,\ldots,q$ need to be put into the projection and, in particular, it is possible to ensure that $Pe \neq 0$ and thus find a descent direction that does not violate those constraints not in the projection, unless optimality has already been achieved. That this is so is a consequence of the equations (2a), (2b), (2c) and (2d). [A point corresponding to such a situation is indicated by y on figure 2 below. That $Pe = 0$ in this case is easy to see since $f_1(y) = f_2(y) = f_3(y)$. We can, in fact, remove $\nabla\phi_3$ from N in this case to obtain descent].

We have noted above that ensuring no active constraints are violated in the corresponding non-linear programming problem is a sufficient condition for a suitable descent direction for M_f.

As we will now indicate, it is not a necessary condition. Furthermore, in such a case, our solution to the problem of finding a descent direction to M_f via the projection matrix P is still suitable.

Suppose our active functions are f_1 and f_2,

Example 2.

$$f_1(x) = x_1^2 + x_2^4$$

$$f_2(x) = (2-x_1)^2 + (2-x_2)^2$$

$$f_3(x) = 2 \exp(-x_1 + x_2).$$

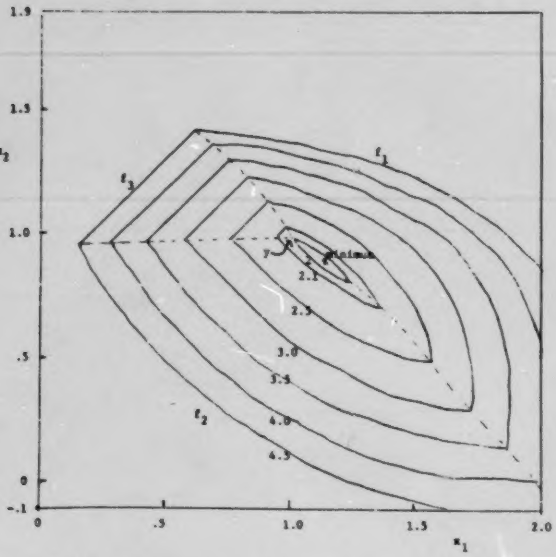


Figure 2. Contours of $M_f(x)$ for Example 2.

both convex functions. That is $q = 2$, $M_f(x) = f_1(x) = f_2(x)$.

Choose d as in (5). The corresponding result to (9) is

$$\phi_i[z+\lambda d^+, x+\lambda d^-] \le \phi_i(z,x) \quad i = 1,2 \quad (10)$$

[since ϕ_i is now concave].

However, by construction

$$e^T d < 0, \text{ i.e. } d^+ < 0$$

and

$$d^T \nabla\phi_i(z,x) = 0 \quad i = 1,2,$$

i.e.,

$$d^+ - [d^-]^T \nabla f_1(x) = 0$$

and

$$d^+ - [d^-]^T \nabla f_2(x) = 0,$$

i.e.,

$$[d^-]^T \nabla f_2(x) = [d^-]^T \nabla f_1(x) = d^+ < 0, \quad (11)$$

or, in other words, locally at least, both f_1 and f_2 have decreased, i.e., d^- is a descent direction for M_f.

Remark. This in itself indicates that the approach we take here is more general than just solving the problem P*.

Furthermore, we remark at this stage that, since a major part of our algorithm depends upon determining P, we can make use of the numerical results available for determining orthogonal projections.

Having determined the search direction we note that minima along a line for a function such as $M_f(x)$ are likely to occur at those points at which the changes in the first derivative are discontinuous [for example, in the linear case this is guaranteed]. Thus, as an initial estimate, the linear search approximates a subset of these points of discontinuity. [Specifically that subset which corresponds to the points of discontinuity given by $f_j(x) = f_\ell(x)$ (where j is one of the active functions at x^k), over all ℓ corresponding to inactive functions at x^k]. If the approximation so determined gives a point where the actual value of M_f is sufficiently lower, then it is accepted - the line search is completed - otherwise a more usual (and more sophisticated) line search is required.

2. The algorithms of Bartels, Charalambous and Conn [3], [5].

The basic ideas behind these two algorithms were those given in section 1 above. As originally presented, the first reference considered only linear functions [actually, piecewise linear since $M_f(x) = \max_i |a_i^T x - \beta_i|$] and the second reference concerned itself with only non-linear functions f_i, - i.e., no attempt was made to improve the efficiency by making use of any linearities present amongst the f_i.

We will attempt to combine the results of both algorithms in one algorithm for the problem P. In addition, differences over the originals will be indicated, as well as relating the results to the work of others in the field, in a later section.

Thus we will use the formulation P* to determine the search direction, but will then apply a related direction directly to the function $M_f(x)$.

In the algorithm, each iteration consists of one, or sometimes two, directions. The first, the one that is always present, is termed the horizontal direction. It tries to maintain constant, the index set of near active functions. Two or more functions are considered near active if they are equal to the present maximum up to a specified tolerance. In addition, it is required that the horizontal direction be such as to decrease M_f. This corresponds to the direction d^- in section 1. The second, the vertical direction, amounts to attempting to satisfy the near active functions exactly by means of linearization. This is done by determining the least squares solution of, in general, an underdetermined system of linear equations. As such this step is closely related, algebraically, to that of determining P and once again is a well-worn path in the field of numerical analysis. The necessity for the vertical direction is given in [5] and details for the computation of both directions are set out below. We note that, in the limit these two directions are orthogonal, hence the terminology.

A linear search follows that incorporates several simple features of the algorithm and numerical results to date indicate that the resulting algorithm is very efficient.

2.1 Notation and Orthogonal Decompositions.

When we want to find the direction of search at a point x^k we set $z = z_k = M_f(x^k)$. By doing so, a near active function in the minimax problem corresponds to a near active constraint in the non-linear programming problem, viz.

$$E(x^k,\varepsilon) = \{i\in[M] \mid M_f(x^k) - f(x^k) < \varepsilon\}$$
$$= \{i\in[M] \mid z_k - f_i(x^k) < \varepsilon\} \qquad (12)$$

We note that $E(x^k,\varepsilon)$ can be considered dependent only on x^k since $z_k = M_f(x^k)$. Let

$$E(x^k,\varepsilon) = NE(x^k,\varepsilon) \cup LE(x^k,\varepsilon), \qquad (13)$$

where $LE(x^k,\varepsilon)$ is the index set for the near-active linear constraints.

$$I(x^k,\varepsilon) = [M]\backslash E(x^k,\varepsilon). \qquad (14)$$

$$\nabla\phi_j(z,x) = [\frac{\partial\phi_j(z,x)}{\partial z}, \frac{\partial\phi_j(z,x)}{\partial x_1}, \ldots, \frac{\partial\phi_j(z,x)}{\partial x_n}]^T \qquad (15)$$

$$= [1, -\nabla f_i(x)]^T. \qquad (16)$$

We note that $\nabla\phi_j(z,x)$ is independent of z.

$$e = [1,0,\ldots,0]^T \quad (= \nabla z). \qquad (17)$$

If the n by k matrix A has independent columns it can be factored into the form

$$A = Q\begin{bmatrix} R \\ 0 \end{bmatrix} = [Q_1 : Q_2]\begin{bmatrix} R \\ 0 \end{bmatrix}, \qquad (18)$$

where $Q = [Q_1 : Q_2]$ is an n by n orthogonal matrix and R is k by k, right triangular, and non-singular. Q_1 denotes the first k columns of Q.

Let $\mathcal{N}(A^T)$ denote the null space of A^T, i.e.,

$$\mathcal{N}(A^T) = \{x \mid A^Tx = 0\}. \qquad (19)$$

The projector on $\mathcal{N}(A^T)$ is given by the matrix

$$Q_2Q_2^T. \qquad (20)$$

Further, suppose h lies in the span of A, i.e.,

$$h = Aw = \sum_{i=1}^{k} w_ia_i, \qquad (21)$$

where $A = [a_1a_2\ldots a_k]$, then w is given by

$$w = R^{-1}Q_1^Th. \qquad (22)$$

Write $w = [w_1\ldots w_k]^T$. In the context of this paper, and in the light of the introduction, we are considering the columns of A to be a subset of the $\nabla\phi_i(z,x)$'s where $i \in E(x,\varepsilon)$.

Thus the direction of (5) is given by

$$d = -Q_2Q_2^Te. \qquad (23)$$

As we shall shortly see, we will actually build up Q by adding columns to A one at a time. Furthermore, in order to eliminate unnecessary re-evaluation at subsequent iterations, those columns that correspond to linear functions $\phi_i(x)$ will be added in first.

At this point let us accept the following convention: if, in a particular context, there will be no ambiguity, we may denote functional expressions dependent upon x and sometimes ε more simply by abandoning one, or often both, of its arguments. Thus $LE(x,\varepsilon)$, $NE(x,\varepsilon)$, $q(x,\varepsilon)$, $E(x,\varepsilon)$, etc., will sometimes be denoted by $LE(x)$, $NE(x)$, $q(x)$, $E(x)$, etc., or sometimes as simply as LE, NE, q or E, etc.

2.2 The Algorithm

Step 0: Set Label = 0, k = 0, vs = 0, $x = x^0$, the starting point, and a value of ε, and εpstop. Set τ_{max}; Note τ_{max} is used in the linear search algorithm. It denotes an upper bound on the admissible stepsize. (Note that Label is used to indicate whether a vertical direction should be taken (Label = 6) or not (otherwise). Similarly vs indicates whether the vertical step was successful (vs = 1) or not (vs = 0)).

Step 1: Set $z_k = M_f(x^k)$. At the point x^k determine the active functions within the specified tolerance. In other words, determine $LE(x^k,\varepsilon)$, $NE(x^k,\varepsilon)$ and hence $E(x^k,\varepsilon)$. If $k \neq 0$, go to Step 3.

Step 2: Determine the projection matrix initially. (This step involves inner iterations).

Step 2a: Set $j = 0$,

$Q = I$, [The $\overline{n+1} \times \overline{n+1}$ unit matrix]

$A = \phi$, (the empty set)

$J_0 = \phi$, $K_0 = \phi$

$q = e$.

J_0 and K_0 are index sets for those linear and non-linear functions, respectively, that are in the projection.

Step 2b: If $LE(x^k,\varepsilon) = \phi$ go to step 2d. Otherwise let $q = Q_2Q_2^Te$, [Recall that Q_2 is the matrix made up of the last (n+1-k) columns of Q] and j = {the i that maximizes

$$\frac{q^T\nabla\phi_i(z,x^k)}{||q||\ ||\nabla\phi_i(z,x^k)||} \;\Big|\; i \in LE(x^k,\varepsilon) - J_0\}.$$

If $q^T\nabla\phi_j(z,x^k) > 0$ go to step 2c, otherwise go to step 2d.

Step 2c: Add a column to A and update Q and R accordingly. Writing $\bar{A} = [A:a]$ where $a = \nabla\phi_j(z,x^k)$ we have

$$\bar{A} = \left[Q\begin{bmatrix} R \\ 0 \end{bmatrix} : a\right] = Q\left[\begin{array}{c:c} R & V_1 \\ \hdashline 0 & V_2 \end{array}\right]$$
$$= [Q_1 : Q_2]\begin{bmatrix} R & V_1 \\ 0 & V_2 \end{bmatrix}$$

where $v_1 = Q_1^T a$, $v_2 = Q_2^T a$.

Now we may take a product of Givens transformations, G_v to transform v_2 into a vector having zero in all the components save the first, where G_v is of the form

$$G_v = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \gamma & \sigma & & \\ & & & \sigma & -\gamma & & \\ & & & & & 1 & \\ & & & & & & \ddots \\ & & & & & & & 1 \end{bmatrix}$$

where γ and $-\gamma$ occupy position v and $v+1$ on the main diagonal. Then

$$\bar{A} = [Q\, G_n \cdots G_{k+1}][G_{k+1} \cdots G_n] \begin{bmatrix} R & v_1 \\ & v_2 \end{bmatrix}$$

$$= \bar{Q} \begin{bmatrix} R & v_1 \\ & v_t \end{bmatrix} = \bar{Q} \begin{bmatrix} \bar{R} \\ 0 \end{bmatrix} \quad (v_t = [\|v_2\|_2, 0, \ldots, 0]^T).$$

Set
$$A = \bar{A},$$
$$Q = \bar{Q},$$
$$R = \bar{R},$$
$$J_0 = J_0 \cup \{j\},$$
$$LE = LE \backslash \{j\}.$$

If the cardinality of J_0 reaches $n+1$, go to step 9 below. Otherwise go to step 2b.

Step 2d: Store Q_1 and Q_2 in $\tilde{Q}_1$ and $\tilde{Q}_2$ respectively.
Store R in $\tilde{R}$.

[In the cases where $LE(x^k, \varepsilon)$ is originally empty the storage of $\tilde{Q}_1$ and $\tilde{Q}_2$ is unnecessary].

Step 2e: If $NE(x^k, \varepsilon) = \phi$ go to step 4. Otherwise set

$$j = \{ i \text{ that maximizes } \frac{q^T \nabla\phi_i(z, x^k)}{\|q\| \, \|\nabla\phi_i(z, x^k)\|} \mid i \in NE(x^k, \varepsilon) - k_0 \}.$$

If $q^T \nabla\phi_j(z, x^k) \le 0$ go to step 4. Otherwise, add $\nabla\phi_j(z, x^k)$ to A and update Q and R as outlined in step 2c above.

Set $K_0 = K_0 \cup \{j\}$,
$NE = NE \backslash \{j\}$.

If the cardinality of $J_0 \cup K_0$ reaches $n+1$ go to step 9, otherwise

$$q = Q_2 Q_2^T e$$

and return to the start of step 2e.

Step 3: Determine the projection matrix (this step involves inner iterations).

Step 3a:

$$Q_1 = \tilde{Q}_1 \quad Q_2 = \tilde{Q}_2 \quad R = \tilde{R}.$$

[Recall that $\tilde{Q}$ is that part of the decomposition that corresponds to just those linear functions $f_i(x)$ included in the projection during the previous iteration. If storage restrictions make it necessary $\tilde{Q}$ may be discarded and recomputed].

Let $q = Q_2 Q_2^T e$. If $LE(x^k, \varepsilon) - J_0 = \phi$, go to step 3d. Otherwise, choose $j \in LE(x^k, \varepsilon) - J_0$, add $\nabla\phi_j(z, x^k)$ to A and update Q and R as above. [Step 2c].

Set
$$J_0 = J_0 \cup \{j\},$$
$$LE = LE \backslash \{j\}$$
$$q = Q_2^T Q_2 e.$$

If $q \neq 0$ go to step 3b.

Step 3c: Drop columns from A and update Q and R accordingly. Suppose

$$\bar{A} = [a_1, a_2, \ldots, a_{j-1}, a_{j+1}, \ldots, a_k],$$

i.e., $\bar{A}$ is A with the jth column deleted. The resulting factorization is of the form

$$\bar{A} = Q \begin{bmatrix} H \\ 0 \end{bmatrix}$$

where H is right Hessenberg and is obtained from R by deleting is jth column. The subdiagonal elements in H, which extend from the new jth (previous (j+1)th) column of R to the last column can be removed by applying an appropriate sequence of Givens transformations, G_v. Thus we may write

$$\bar{A} = Q[G_j \cdots G_{k-1}][G_{k-1} \cdots G_j] \begin{bmatrix} H \\ 0 \end{bmatrix}$$
$$= \bar{Q} \begin{bmatrix} \bar{R} \\ \hdashline 0 \end{bmatrix} .$$

$q = 0$ implies that e lies in the span of the columns of A, i.e.,

$$Aw = e.$$

Since $A = Q \begin{bmatrix} R \\ 0 \end{bmatrix}$ the vector w is given by

$$w = R^{-1} Q_1^T e,$$

i.e., set $y = Q_1^T e$ and solve

$$Rw = y.$$

If $w \ge 0$, stop, x^k is optimal. Otherwise, choose j such that $w_j < 0$. [Typically one chooses j such that w_j is most negative]. and drops the corresponding a_j from A, updating the Q and R as outlined above.

Set
$$J_0 = J_0 \backslash \{j\},$$
$$LE = LE + \{j\}.$$

Step 3d: As for step 2d above.

Step 3e: As for step 3d above. [Except that the last statement is replaced by "return to the start of step 3e"].

Step 4: Set $d = -q$.

Let $A = \{i_1 \dots i_j\}$ denote $J_0 \cup K_0$.

Step 5: Check if optimum is reached.
If $||d|| < \varepsilon$stop and Label $\neq 6$ or if $\varepsilon < \varepsilon$pstop, stop.

Step 6: Linear Search

The linear search is done directly on the minimax function. Let d^- be the direction d of step 4 with the first component deleted (remember that the first component of d (d^+) corresponds to the z and d^- corresponds to $x \in R^n$). Then d^- has the property that it will try to decrease the subset of the active functions at the point x^k, $f_{i_1}(x)$, $f_{i_2}(x)$, ..., $f_{i_j}(x)$, (i.e., those we put into the projection) by the same amount. (Thus decreasing $M_f(x)$, since, by construction, the remaining active functions at the point x^k will locally decrease along d^-, as this is the basis on which the projection matrix was determined). Thus at the point x^k we move downhill along the valley defined by the minimax functions $f_{i_1}(x), f_{i_2}(x)$, ..., $f_{i_j}(x)$. Note that if we consider only one function in P, say the function $f_{i_1}(x)$, then $d^- = -\nabla f_{i_1}(x)$, which is the steepest descent direction for the function $f_{i_1}(x)$.

Determine $\tau > 0$ such that

$$\text{Max } f_i(x^k + \tau d^-),\ i \in [M]$$

is minimized. Call this τ, τ_{opt}

For details of how this is done and whether exact minimization is required, see the subalgorithm "The Linear Search Algorithm" below. Put

$$x^k \leftarrow x^k + \tau_{opt}\, d^-.$$

Step 7: Decision as to whether to do the vertical step or not. The vertical direction amounts to attempting to make the near active functions exactly equal, and by doing so an effort is made to get exactly on the line of the discontinuous derivatives, which is very desirable when we are close to the solution. We expect that we will be close to the solution either when the active functions remain the same at each iteration or if we are trying to consider more than n functions in the projection. [Actually, even if we are not near the solution, should either of these two phenomena occur we might need to "reach the valley floor" in order to move away from the current situation]. Algorithmically, if the number of active functions have not changed in three consecutive iterations, $K_0 \neq \emptyset$, and $||d|| < .1$, or if Label = 6 (see step 9) go to step 8. Otherwise, set

$$x^{k+1} = x^k$$
$$k \leftarrow k+1$$

and go to step 1.

Step 8: The Vertical Step [involves inner iterations].

Set $z_k = M_f(x^k)$. Determine $E(x^k, \varepsilon)$. (Note that x^k is the point obtained from the horizontal step).

Step 8a: Set $Q_1 = \tilde{Q}_1$, $Q_2 = \tilde{Q}_2$, $R = \tilde{R}$.

Step 8b: As for step 2e above [except that the last statement is replaced by "return to the start of step 8b" and "go to step 4" is replaced by "go to step 8c"].

Step 8c: $v(x^k, \varepsilon) = -Q_1 R^{-T} \Phi$

where $\Phi = (\phi_{i_1} \dots \phi_{i_j})^T$.

Put $x_{temp} = x^k + v$ (with 1st component missing)
$= x^k + v^-$.

If

$$\underset{i \in [M]}{\text{Max}}\, f_i(x_{temp}) < \underset{i \in [M]}{\text{Max}}\, f_i(x^k),$$

set $vs = 1$,

$$x^{k+1} = x_{temp},\ k \leftarrow k+1,$$

Label = 0 and go to step 1. Otherwise set vs = 0,

$$x^{k+1} = x^k,\ k \leftarrow k+1$$

and go to step 1.

(Note that $f_{i_\ell}(x^k) + \nabla f_{i_\ell}^T(x^k) v^- = M_f(x^k) + v^+ + \varepsilon_2$ where $|\varepsilon_2| < \varepsilon$, $\ell = 1, 2, \dots, j$, and v^+ is the first component of $v(x^k, \varepsilon)$. Therefore the linearized active minimax functions at x^k will be within ε at the point $x_{temp} = x^k + v^-$).

Step 9: Trying to put $(n+1)$ constraints in the projection, i.e.,

$$|J_0 \cup K_0| = n+1.$$

$|J_0 \cup K_0| = n+1$ means one of two possibilities, either a) we have reached the neighbourhood of the optimum, or b) we have constraints considered active that actually are not. This situation is handled in two ways. Firstly, by ensuring that we take a vertical step and secondly, by reducing ε. Algorithmically, set label = 6, put $\varepsilon = \varepsilon/10$ and go to step 8.

The Linear Search Algorithm

Step 0: If vertical step was successful (i.e., if vs = 1), go to step 4.

Step 1: Estimate any new function to become active. We consider all inactive constraints ϕ_j and estimate, in turn, the stepsize to make each ϕ_j zero. Hence we calculate

$$\tau_j = \frac{\phi_j(z_k, x^k)}{\nabla \phi_j^T(z, x^k) d}, \quad j \in I(x^k, \varepsilon). \tag{24}$$

In other words, if ϕ_j was linear we calculate τ_j so that $\phi_j((z_k,x^k) + \tau_j d) = 0$ and in general we linearize the ϕ's. Let $f_m(x^k) = M_f(x^k)$ for some m, $f_j(x)$ an inactive function at the point x^k and τ_j as calculated by 24. Then we want the linear approximation of the functions f_m and f_j about the point x^k to be intersected at the point $x^k + \tau_j d^-$. By construction q is orthogonal to $\nabla(z - f_m(x))$ at the point x^k, i.e.,

$$\nabla f_m(x^k)^T d^- = d^+. \tag{25}$$

Also, by construction of τ_j

$$z_k - f_j(x^k) + \tau_j[1, -\nabla f_j(x^k)^T]\begin{bmatrix} d^+ \\ \hdashline d^- \end{bmatrix} = 0.$$

By using the fact that $z_k = M_f(x^k)$ we have

$$\begin{aligned} f_j(x^k) + \tau_j \nabla f_j(x^k)^T d^- &= z_k + \tau_j d^+ \\ &= M_f(x^k) + \tau_j d^+ \\ &= f_m(x^k) + \tau_j d^+ \qquad (26) \end{aligned}$$

From 25

$$f_m(x^k) + \tau_j \nabla f_m(x^k)^T d^- = f_m(x^k) + \tau_j d^+ \tag{27}$$

Comparing (26) and (27) we can see that we have the desired result.

Step 2: Omitting unlikely values of τ_j, estimate the optimum, τ_{opt}, by linearizing the minimax function about x^k. For $j \in I(x^k,\varepsilon)$ do the following. If $\tau_j < 0$ or $\tau_j > \tau_{max}$ neglect it as inadmissible. Otherwise calculate

$$\hat{f}_{ij} \triangleq f_i(x^k) + \tau_j \nabla f_i(x^k)^T d^-, \quad i \in [M]$$

where

$$\nabla f_i(x) = \left[\frac{\partial}{\partial x_i} f_i(x), \ \ldots, \ \frac{\partial}{\partial x_i} f_i(x)\right]^T$$

Put

$$\hat{F}_j = \max_{i \in [M]} \hat{f}_{ij}.$$

Now, determine ℓ such that

$$F_\ell = \text{Min } \hat{F}_j$$

$$j \in I(x^k,\varepsilon) \backslash \{j | \tau_j < 0 \text{ or } \tau_j > \tau\text{max}\}.$$

Put $\tau_{opt} = \tau_\ell$.

Step 3: Determine if τ_{opt} is acceptable. Calculate the true minimax value at

$$\hat{x}^k = x^k + \tau_{opt} d^-.$$

If this new value is an improvement over the old value, set $x^k = \hat{x}^k$. Otherwise go to step 4.

Step 4: Use cubic line search on the maximum of the functions taking τ_{opt} as an upper bound, to obtain the new x^k, if τ_{opt} is available from step 3 [i.e., if vs = 0].

2.3 Some remarks on the algorithm

i) As is noted in [4] the QR decomposition and its update when a column is added or deleted from A, arises with such frequency that a wealth of techniques have been developed to handle the situation. The method set out here corresponds to that in [11]. It is used presently solely because it is the simplest to explain. For alternative methods, more details, and references, see [4].

ii) In practise we do not use an exact cubic linear search but merely ask for sufficient improvement in the minimax value.

iii) If in step 7 of the main algorithm we decided to do the vertical step and it was successful, we dispense with the estimation of τ_{opt} as above and merely do the cubic search. The motivation for this is as follows. The estimates for τ_i are based on the surmise that some new function will become active whereas the vertical step is based on the assumption that this will not be the case.

iv) In the above algorithm we have assumed that the columns given by $\nabla\phi_i(x,\varepsilon)$, $i \in LE(x,\varepsilon)$ are linearly independent. The situation in which the columns are dependent corresponds precisely to degeneracy in linear programming. As was done in [4], we will regard degeneracy as a condition which can be cleared up through the use of negligible perturbations of the data, and we shall ignore the linearly-dependent case.

v) Under some fairly natural assumptions the above algorithm can be proved to be convergent. The proofs are somewhat lengthy and are given elsewhere [5]. In the case of solely linear functions, finite convergence can be proved. [See [4]].

2.4 Relationship of the above with other algorithms

i) The algorithm differs from that given in [5] in that it is a stable implementation. Furthermore, whenever possible, advantage is taken of any linear functions that might be active. This is done by preferentially entering the linear functions into the projection and storing that part of the decomposition (QR) that corresponds to the linear functions alone. Thus, instead of recomputing the entire decomposition at each iteration, the linear part can be updated.

ii) As was remarked above, in the purely linear case, the algorithm 2.2 does not, quite, reduce to the algorithm of [4]. This is because in that algorithm $M_f(x) = \text{Max}_i |f_i(x)|$ and, in addition, the linear search differs from that above. Thus the corresponding P* is

$$\text{Min } z$$

subject to

$$z - |f_i(x)| \geq 0, \quad i \in [M]. \tag{28}$$

It is apparent that the above problem is equivalent to

$$\operatorname{Min} z$$

subject to,

$$\left.\begin{array}{l} z - f_i(x) \geq 0, \\ z + f_i(x) \geq 0, \end{array}\right\} i \in [M] \qquad \underline{P^{*\prime}}$$

with corresponding

$$\underset{x}{\operatorname{Min}} \ \underset{i \in [M]}{\operatorname{Max}} \{f_i(x), - f_i(x)\} \qquad \underline{P'}$$

If one formulates the problem this way, eliminates the vertical step on account of it being superfluous in the purely linear case, then, excepting the linear search, one has, essentially, the algorithm of [3]. For completeness I point out that they use fast Givens without square roots in their decomposition of A instead of the QR given above. (The elements of fast Givens are given in [13]).

There are three linear searches mentioned in [3]. Besides the complexity introduced by the absolute modulus function, none of the methods correspond exactly to that given in algorithm 2.2, (henceforth referred to as LSA(2.2)). For simplicity we will describe equivalent forms for the original problem P.

__Linear Search Algorithm A__. (LSA(A)).

As for LSA(2.2) but determine ℓ such that τ_ℓ is the smallest positive τ_j, $j \in I(x^k, \varepsilon)$, [i.e., locate the first break-point reached in the direction d^- of the piecewise linear function M_f on leaving x^k].

__Linear Search Algorithm B__. (LSA(B)

__Step 1__: As for step 1 of LSA(2.2).

__Step 2__: Put τ_ℓ as the smallest positive τ_j, $j \in I(x^k, \varepsilon)$.

__Step 3__: [Determine whether M_f is still decreasing in the direction d^-].

If $d^- \nabla f_\ell(x^k) > 0$, terminate the linear search algorithms with $\tau_{opt} = \tau_\ell$.

Otherwise it is necessary to make the following redefinitions

$$x^k \leftarrow x^k + \tau_\ell d^-$$
$$z^k \leftarrow M_f(x^k + \tau_\ell d^-)$$

and return to step 1.

__Linear Search Algorithm C__. LSA(C)

This algorithm was a mixture of LSA(A) and (B). Specifically it used LSA(A) until such time as $q = 0$ in step 3b of algorithm 2.2, above. Then in step 6 it used LSA(B). In subsequent iterations one returned to LSA(A) for step 6 until $q = 0$ again in step 3b, and so on.

Schematically we have the following situation for the linear search.

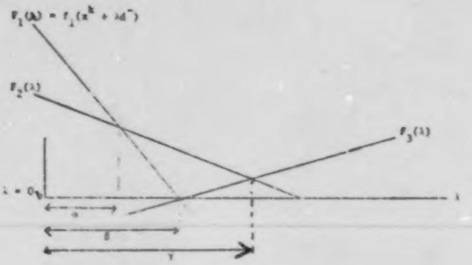


__Figure 3__.

α corresponds to τ_{opt} from LSA(A)

β " " " " LSA(2.2)

γ " " " " LSA(B)

Using LSA(A), with the exception of, possibly, the first $n+1$ iterations, each of the points x^k were vertices for the (linear programming) problem $P^{*\prime}$ (or at least satisfied a maximal number of active constraints in case the problem was rank deficient). It is for this reason that they preferred strategy LSA(A). However, in the case where the $f_i(x)$'s are non-linear, essentially, as is usual in non-linear programming, one is looking only for a sufficient decrease in the objective function, $M_f(x)$. Strategy LSA(B) is undesirable in that it requires too much additional computation.

iii) The algorithm of [10] corresponds exactly to the algorithm above using LSA(A) (except that Cline is restricted to Haar-condition problems) starting at a vertex to the (linear programming) problem P'. In addition to being able to handle non-Haar problems, the treatment given above is more flexible. In particular the addition of constraints can be handled in a natural way that is particularly simple in the case of linearly constrained problems.

iv) Perhaps the best known algorithm for handling problems of type (28) in the linear case is that of Barrodale and Phillips [2]. They, in fact, add an additional linear constraint, viz.

$$z > 0$$

to $P^{*\prime}$ and then solve the dual linear program. The structure of the problem enables them to condense the resulting tableau by supressing all but m of the columns. There is some suggestion that in practice the dual form, at least for linear problems, is the best approach to the ℓ_∞ problem. However, to the best of the present author's knowledge, there is, as yet, no well-documented evidence to support this assertion. In particular, let us formulate the algorithm of Barrodale and Phillips in terms of the primal. The algorithm consists of three stages. More specifically, suppose the problem under consideration is

$$\underset{x}{\operatorname{Min}} \left\{ ||b - Ax||_\infty = \underset{1 \leq i \leq m}{\operatorname{Max}} \left| b_i - \sum_{j=1}^{n} a_{ij} x_j \right| \right\}$$

and the rank of A is k.

In stage 1 one chooses the residual of maximum absolute value and essentially, uses the steepest descent direction to bring its value to zero. One then updates the residual values choosing that which is now maximal in absolute value and bringing it down to zero, whilst still holding those residuals that are already at zero at zero. One then iterates until k residuals are at zero.

This is readily accomplished by means analogous to those above for algorithm 2.2. For example, suppose

$$b_i - \sum_{j=1}^{n} a_{ij}x_j = 0 \quad i = 1,2,\dots,q,$$

and $|e_r| = |b_r - \sum_{j=1}^{n} a_{rj}x_j|$ is maximal.

Define P as in (3) with $N = [a_1 \dots a_q]$ where $a_j = [a_{j1} \dots a_{jn}]^T$.

The required descent direction is given by $d = Pg$ where $g = a_r \operatorname{sign}(e_r)$. ($\operatorname{sign}(e_r) = 1$ if $e_r > 0$, and -1 if $e_r < 0$ and 0 otherwise).

Since in stage 1, typically one increases the number of residuals with value zero by 1, the orthogonal decompositions, plus corresponding updates, are applicable here. Stage 1 is terminated when k residuals have the value zero.

Stage 2 consists of increasing all the k residuals (in absolute value) simulateously, until $k+1$ residuals have the same absolute value. Stage 2 involves only one iteration.

Stage 3 is equivalent to the well-known exchange algorithm for linear minimax approximation (see for example [14]).

Again Stages 2 and 3 can be accomplished with linear algebra analogous to that of algorithm 2.2.

The above is only a very brief outline of the algorithm of Barrodale and Phillips. The details of the condensed simplex implementation are given in [1].

In the light of the above comments one is able to see that the "dual" formulation of Barrodale and Phillips can be realized in a stable way by a direct formulation as for algorithm 2.2. It then seems natural to ask the question that if the dual approach to the minimax problem is superior to the direct approach, when is a dual approach indirect? Furthermore, Barrodale and Phillips claim that the main feature of their algorithm is that it enters stage 3 with a value close to optimal. However, if one compares how one arrives at stage 3, in terms of the primal problem, to the analogous situation in algorithm 2.2 [i.e., when one reaches a vertex (or point where $q = 0$, more generally)] the apparent superiority of the "indirect" or "dual" methods appears surprising.

v) Much of the ideas of [17] are in fact equivalent to that of the horizontal direction of algorithm 2.2. However, the vertical direction is dispensed with at the cost of having to reduce the tolerance to which functions are considered active, to zero. This makes ultimate convergence difficult (cf. [7] and [9] for a similar result). Furthermore the numerical aspects of the Zangwill algorithm are not considered by him. In particular, no linear search suitable for minimizing $M_f(x)$ is given. Finally, Zangwill introduced separate cases dependent on whether a certain matrix is of full, or almost full, ranks. Such a differentiation of cases is numerically undesirable and does not occur in algorithm 2.2.

2.5 Some additional remarks and conclusions.

As is noted above, and in [4] for the linear case, the addition of constraints to problem P can be handled in a natural way. This is an immediate consequence of the fact that algorithm 2.2 is based on projections.

One might also remark that since the projections in algorithm 2.2 are orthogonal, at least in terms of the non-linear programming problem P^*, the iterations consist of steepest-descent-like iterations in certain subspaces. It might be suggested that one might be able to determine Newton or Quasi-Newton type iterations by using non-orthogonal type projections (see for example [15] and [12] for parallels in non-linear programming). However, since the objective function (z) in P^* is linear, the straightforward non-orthogonal projection will not do the job. However, consider the case of just one function $f_1(x)$, say, active. Algorithm 2.2 gives $-\nabla f_1(x)$ for d^- - i.e., steepest descent. However, for general non-linear f the corresponding Newton direction $-G_f^{-1}(x)f(x)$ is well defined [where G_j denotes the Hessian of f; here assumed invertible]. In other words, a second-order type generalization is not obvious, even assuming it is desirable.

The above is meant to be a summary of one (direct) approach to minimax problems via projections. Some attempt has been made to relate the approach to other algorithms designed for the same problem. It is also hoped that the results reported here indicate that there is still work to be done in the area with suggestions as to some avenues of future research.

Numerical results are not reported here. Suffice it to say that numerical results have been carried out in the context of applications, general non-linear and linear problems. They are reported in detail in references [5], [3] and [6].

Acknowledgements.

Much of what appears in this paper is based on earlier work carried out with two colleagues, R. Bartels and C. Charalambous. In addition, R. Bartels impressed on me the importance of numerical linear algebra in the context of mathematical programming and C. Charalambous introduced me to the minimax problem. To both I am grateful.

This research was supported in part by grant number A8639 from the National Research Council of the Canadian Government.

References

[1] I. Barrodale and C. Phillips, "An Improved Algorithm for Discrete Chebyshev Linear Approximations", Proc. 4th Manitoba Conf. on Numerical Mathematics, University of Manitoba, Winnipeg, Canada, 1974, pp. 177-190.

[2] I. Barrodale and C. Phillips, "Algorithm 495: Solution of an Overdetermined System of Linear Equations in the Chebyshev Norm", ACM Transactions on Mathematical Software, Vol. 1, No. 3, September 1975, pp. 264-270.

[3] R.H. Bartels A.R. Conn and C. Charalambous, "Minimization Techniques for Piecewise Differentiable Functions: The ℓ_∞ Solution to an Overdetermined Linear System", Research Report CORR 76/30, September 1976.

[4] R.H. Bartels, A.R. Conn and J.W. Sinclair, "Minimization Techniques for piecewise differentiable functions - the ℓ_1 solution to an over-determined linear system", SIAM J. Num. Anal. (to appear)

[5] C. Charalambous and A.R. Conn, "An efficient method to solve the minimax problem directly", Dept. Combinatorics and Optimization, Univ. Waterloo, Ont. Canada. Rep. CORR 74/19.

[6] C. Charalambous and A.R. Conn, "Optimization of Microwave Networks", IEEE Trans. Microwave Theory Tech. (1975), 834-838.

[7] A.R. Conn, "Constrained optimization using nondifferentiable penalty function", SIAM J. Num. Anal. vol. 10, pp. 760-784, 1973.

[8] A.R. Conn, "Projection Matrices - a A fundamental concept in optimization" in Proceedings of the 7th Annual Conference on Modelling and Simulation, Pittsburgh, April 26-27, 1976.

[9] A.R. Conn and T. Pietrzykowski, "A Penalty Function Method converging directly to a Constrained Optimum", SIAM J. Num. Anal. Vol. 14 ,No. 2 (1977).

[10] A.K. Cline, "A Descent Method for the Uniform Solution to Overdetermined Systems of Linear Equations", SIAM J. on Num. Anal. Vol. 13, No. 3, June 1976, pp. 293-309.

[11] P.E. Gill, G.H. Golub, W. Murray and M.A. Saunders, "Methods for Modifying Matrix Factorizations", Mathematics of Computation, Vol. 28, No. 126, April 1974, pp. 505-535.

[12] D. Goldfarb and L. Lapidus, "Conjugate gradient methods for non-linear programming problems with linear constraints", I and E.C. Fundamentals, 7 (1968), pp. 142-151.

[13] R.J. Hanson and C.L. Lawson, "Solving least squares problems", Prentice-Hall, Englewood Cliffs, N.J. 1974.

[14] M.J. D. Powell, "The minimax solution of linear equations subject to bounds on the variables", AERE Harwell, Oxfordshire, England. Rep. CSSII, December 1974.

[15] J.B. Rosen, "The gradient projection method for non-linear programming. Part I: Linear constraints", J. Soc. Indust. Appl. Math. 8 (1960), pp. 181-217.

[16] A.D. Waren, L.S. Lasdon and D.F. Suchman, "Optimization in engineering design", Proc. IEEE, vol. 55, pp. 1885-1897, Nov. 1967.

[17] W.I. Zangwill, "An algorithm for the Chebyshev Problem - with an application to Concave Programming", Manag. Sci. Vol. 14 #1 (1967), 58-78.

[18] G. Zoutendijk, "Methods of Feasible Directions", Elsevier Publ. Co. Amsterdam, 1960.

NUMERICAL ASPECTS OF TRAJECTORY ALGORITHMS FOR NONLINEARLY CONSTRAINED OPTIMIZATION

Walter Murray
National Physical Laboratory, Teddington, England

Margaret H. Wright
Stanford University

Abstract

This paper discusses two algorithms for nonlinearly constrained optimization. These algorithms -- the penalty and barrier trajectory algorithms -- are based on an examination of the trajectories of approach to the solution that characterize the quadratic penalty function and the logarithmic barrier function, respectively. Although closely related in principle, the two algorithms display important differences in their implementation as well as in the properties of the generated iterates. The discussion will emphasize the numerical aspects of implementation of the trajectory algorithms, with particular attention to the choice of reliable methods for carrying out the required computations.

1. Introduction

The problem to be considered in this paper is the following:

$$\text{P1:} \qquad \text{minimize} \quad F(x), \qquad x \in E^n$$

$$\text{subject to} \quad c_i(x) \geq 0, \; i = 1,2,\ldots,m,$$

where $F(x)$ and $\{c_i(x)\}$ are prescribed nonlinear functions. The function $F(x)$ is usually termed the "objective function" and the set $\{c_i(x)\}$ is the set of "constraint functions". It will be assumed for simplicity that F and $\{c_i\}$ are twice continuously differentiable, although the methods to be discussed will cope with occasional discontinuities.

The solution of P1 will be denoted by x^*. In all problems to be considered, the first- and second-order Kuhn-Tucker conditions are assumed to be satisfied at x^*, so that there exists a vector λ^* of non-negative Lagrange multipliers, corresponding to the active constraints at x^*, satisfying

$$g(x^*) - \hat{A}(x^*)\lambda^* = 0 \qquad (1)$$

where $g(\cdot)$ is the gradient of F, and the columns of $\hat{A}(\cdot)$ are the gradients of the constraints active at x^*.

It is customary to state problem P1 with two kinds of constraints -- equality constraints (of the form $c_j(x) = 0$), as well as the inequalities given above in P1. This distinction will not be made during the present discussion, in order to avoid the introduction of additional notation. The treatment of inequality constraints is typically more complicated than that of equality constraints, and the algorithms to be discussed can deal in an obvious way with equality constraints.

Because the problem P1 can not, in general, be solved explicitly, a popular approach during the past decade has been to transform P1 into a sequence of unconstrained minimization problems. The most common such transformation has been effected by the use of penalty and barrier function methods, which are discussed at length in, for example, Fiacco and McCormick (1968) and Ryan (1974). This paper will not review these methods, but their properties are critical in the derivation of the algorithms to be discussed.

The quadratic penalty function corresponding to the problem P1 is defined by

$$P(x,\rho) \equiv F(x) + \frac{\rho}{2} \sum_{i \in I} (c_i(x))^2 , \qquad (2a)$$

where I is a subset of the indices $\{1,2,\ldots,m\}$ (usually, the set of constraints whose values are less than a small positive number), and ρ is a positive scalar termed the "penalty parameter". The quadratic penalty function may also be written as:

$$P(x,r) \equiv F(x) + \frac{1}{2} r^{-1} \sum_{i \in I} (c_i(x))^2 , \quad (2b)$$

where $r = 1/\rho$. Let $x^*_P(r)$ denote an unconstrained minimum of $P(x,r)$.

The logarithmic barrier function corresponding to the problem P1 is given by:

$$B(x,r) \equiv F(x) - r \sum_{i=1}^{m} \ln(c_i(x)) , \quad (3)$$

where r is a positive scalar termed the "barrier parameter". The logarithmic barrier function is defined only at points that strictly satisfy all of the problem constraints. Let $x^*_B(r)$ denote an unconstrained minimum of $B(x,r)$.

Under certain mild conditions, there exists $\hat{r} > 0$ such that for $r < \hat{r}$, $x^*_P(r)$ and $x^*_B(r)$ are continuous functions of r, and:

$$\lim_{r \to 0} x^*_P(r) = x^* ;$$

$$\lim_{r \to 0} x^*_B(r) = x^* .$$

Although penalty and barrier function methods display several good features, they suffer from certain theoretical and numerical defects -- in particular, both require the solution of a theoretically infinite sequence of unconstrained problems. Furthermore, in practice each successive unconstrained sub-problem is more difficult to solve, because the Hessian matrices of the penalty and barrier functions become increasingly ill-conditioned as r approaches zero, and are singular in the limit (see Murray, 1969a).

The continuous lines of minima in E^n defined by $x^*_P(r)$ and $x^*_B(r)$ are termed the "penalty trajectory" and "barrier trajectory" of approach to x^*, respectively. The analysis of these trajectories, and a detailed description of the trajectory algorithms, have been given elsewhere (Murray, 1969a,b; Wright, 1976; Murray and Wright, 1976b); for the purposes of the present discussion, only a brief description of the underlying motivation will be stated.

The trajectory algorithms are based on using the properties of the trajectories to generate a sequence of iterates that lie in a neighborhood of the appropriate trajectory, in order to mimic the approach to x^* of the iterates from a penalty or barrier function method. Because it is possible to characterize a step toward the penalty or barrier trajectory without assuming that the current iterate is in a close neighborhood of x^*, the derivation of the trajectory methods does not display a stringent dependence on properties that hold only in such a neighborhood. Moreover, it is not necessary for any of the iterates to lie exactly on the trajectory (as in a penalty or barrier function method).

At each iteration of a trajectory method, the search direction is computed as a step toward some point on the desired trajectory. The particular point to be aimed for depends on the current value of the penalty or barrier parameter. The solution x^* is also on the trajectories, and the target point will ultimately become arbitrarily close to x^* as the algorithms converge. The penalty or barrier parameter may be adjusted at each iteration of the trajectory methods; however, the choice of the parameter value is not critical, since a step to a neighborhood of a point on the trajectory corresponding to $\bar{r}$ is also in the neighborhood of a point corresponding to $(1+\varepsilon)\bar{r}$, where ε is small. The numerical procedure for determining the search direction in both algorithms is well-posed, and the approach to the limit of the penalty or barrier parameter does not cause any ill-conditioning.

This paper will emphasize some numerical aspects of implementation of the trajectory methods, with particular attention to the choice of reliable procedures for carrying out the required computations. The emphasis on the details of implementation is deliberate; even within an algorithm that has been designed from the outset to be robust, additional safeguards are necessary to protect against failure or illogical results when the underlying assumptions are not satisfied.

2. Description of Trajectory Algorithms

Only certain key aspects of implementation of the trajectory methods have been selected for discussion in Section 3. Accordingly, the descriptions given here of the penalty and barrier trajectory algorithms are slightly abbreviated, and do not contain all the computational details. A complete description of both algorithms is given in Murray and Wright (1976b).

2.1. Penalty Trajectory Algorithm

2.1.1. Properties of the search direction

For the penalty trajectory algorithm, at each iteration the search direction, p, is (ideally) constructed as the solution of the following quadratic program:

$$\text{QP1:} \qquad \min \; \frac{1}{2} p^T Sp + p^T g$$

$$\text{subject to} \quad \hat{A}_p^T = -\hat{c} - \frac{1}{\rho}\lambda \quad ,$$

where $\hat{c}$ denotes the vector of constraints currently considered "active"; $\hat{A}$ is a matrix whose columns are the gradients of the active constraints; λ is an estimate of the Lagrange multiplier vector; ρ is the current value of the penalty parameter; g is the gradient of F; and S is a matrix that approximates the Hessian of the Lagrangian function.

Let Y be a matrix whose columns form an orthogonal basis for the range of the columns of $\hat{A}$, and let Z be a matrix whose columns form an orthogonal basis for the corresponding null space, i.e.,

$$\hat{A}^T Z = 0 \; ,$$

$$Z^T Z = I \; .$$

If $\hat{A}$ has full column rank ($\leq n$), and if the matrix $Z^T SZ$ is positive definite, then the solution of QP1, p^*, can be uniquely expressed as the sum of two orthogonal components:

$$p^* = Yp_R + Zp_N \; .$$

For sufficiently large ρ, the search direction p^* so constructed will always be a descent direction for the quadratic penalty function; the step to be taken along the search direction is then chosen to achieve an acceptable decrease in the penalty function.

2.1.2. Calculation of the search direction

At the beginning of the k-th iteration of the penalty trajectory algorithm, the following vectors and matrices are assumed to be available:

- $x^{(k)}$, an approximation to x^*;
- $c^{(k)}$, the vector of values of $\{c_i(x)\}$ evaluated at $x^{(k)}$;
- $g^{(k)}$, the gradient vector of $F(x)$ evaluated at $x^{(k)}$;
- $A^{(k)}$, the matrix whose columns are the gradients of $\{c_i(x)\}$ evaluated at $x^{(k)}$;
- $S^{(k)}$, an approximation to the Hessian matrix of the Lagrangian function at $x^{(k)}$.

The procedures followed during the k-th iteration to compute the next iterate are:

(1) An "active set" of constraints is determined, containing $\leq n$ elements (see Section 3.1 for a discussion of the case where the active set contains more than n elements). The vector of active constraint values will be denoted by $\hat{c}$, and the matrix whose columns are the gradients of those constraints will be denoted by $\hat{A}$.

(2) Factorize $\hat{A}$ such that

$$Q\hat{A} = \begin{bmatrix} R \\ --- \\ 0 \end{bmatrix} , \qquad Q^T Q = I \; .$$

where R is an upper triangular matrix. Define the matrices Y and Z by partitioning Q as:

$$Q = \begin{bmatrix} Y^T \\ ------ \\ Z^T \end{bmatrix} \; .$$

(3) Determine an estimate, λ, of the Lagrange multiplier vector. If $\hat{A}$ has full solumn rank, λ is the least-squares solution of $\min \|\hat{A}\lambda - g^{(k)}\|_2^2$, and is given by the solution of the triangular

system:

$$R\lambda = Y^T g^{(k)} .$$

If A is rank-deficient, the vector λ will be taken as the minimum-length least-squares solution; it is obtained by extending the factorization of Step (2) to:

$$Q\hat{A}V = \left[\begin{array}{c|c} \bar{R} & 0 \\ \hline \multicolumn{2}{c}{0} \end{array}\right] , \qquad (4)$$

where $\bar{R}$ is a non-singular upper triangular matrix and V is an orthogonal matrix (see Peters and Wilkinson, 1970, for further details).

(4) Determine an appropriate value of the penalty parameter, ρ (Murray and Wright, 1976b).

(5) Compute the vector p_R, as follows. If $\hat{A}$ has full rank, p_R is obtained by solving the linear system:

$$\hat{A}^T p = \hat{A}^T Y p_R = R^T p_R = -\hat{c} - \frac{1}{\rho}\lambda . \qquad (5)$$

In this way, the direction Yp_R satisfies the linear equality constraints of QP1. If $\hat{A}$ is rank-deficient, p_R is a least-squares solution of (5), computed using the complete orthogonal factorization (4); the linear constraints of QP1 will then not be exactly satisfied.

(6) Determine the modified Cholesky factorization of the matrix $Z^T S^{(k)} Z$. With this procedure, the matrix $Z^T S^{(k)} Z$ is augmented (if necessary) by a positive diagonal matrix, E, chosen to make $(Z^T S^{(k)} Z + E)$ strictly (numerically) positive definite. Let LDL^T be the computed factorization, so that:

$$LDL^T = Z^T S^{(k)} Z + E ,$$

where E is identically zero if $Z^T S^{(k)} Z$ is sufficiently positive definite (Gill and Murray, 1972a).

(7) Determine the vector $\bar{p}_N$ by solving

$$LDL^T \bar{p}_N = -Z^T g^{(k)} .$$

Test whether:

$$\|\bar{p}_N\| \le M\|p_R\| \quad \text{and} \quad \|p_R\| \le M\|\bar{p}_N\| , \qquad (6)$$

for M a reasonably large positive number (say, 1,000).

(a) If the test (6) is satisfied (as it almost always is in practice), compute p_N by solving:

$$LDL^T p_N = -Z^T(g^{(k)} + S^{(k)} Y p_R) ;$$

then form the search direction as:

$$p = Yp_R + Zp_N .$$

(b) If the test (6) is not satisfied, the two orthogonal portions of the search direction are not well-scaled, and the following re-scaling procedure is used to adjust for the imbalance.

If $\|\bar{p}_N\| > M\|p_R\|$, define a scaling factor β_1 as

$$\beta_1 = \frac{M\|p_R\|}{\|\bar{p}_N\|} ,$$

and let

$$p = Yp_R + \beta_1 Z\bar{p}_N ;$$

otherwise, define a scaling factor β_2 as

$$\beta_2 = \frac{M\|\bar{p}_N\|}{\|p_R\|} ,$$

and let

$$p = \beta_2 Yp_R + Z\bar{p}_N .$$

(8) Determine a step length, α, that generates an acceptable reduction in the penalty function $P(x, \rho)$, using a safeguarded cubic or parabolic step length algorithm (e.g., the procedure described in Gill and Murray, 1974). Special care must be exercised in the step length algorithm to avoid difficulties if $P(x, \rho)$ is unbounded below along the given search direction (see Section 3.5.1).

(9) Set $x^{(k+1)}$ to $x^{(k)} + \alpha p$, and return to Step (1).

2.2. Barrier Trajectory Algorithm

2.2.1. Properties of the search direction

The search direction of the barrier trajectory algorithm is (ideally) constructed as the solution of the following quadratic program:

$$\text{QP2:} \qquad \min \frac{1}{2} p^T Sp + p^T g$$

$$\text{subject to } \hat{A}^T p = d \ ,$$

where $d_i = -\hat{c}_i + r/\lambda_i$; $\hat{c}$ denotes the vector of constraints currently considered "active"; $\hat{A}$ is a matrix whose columns are the gradients of the active constraints; λ is an estimate of the Lagrange multipliers corresponding to the active constraints; r is the current value of the barrier parameter; g is the gradient of F; and S is an approximation to the Hessian of the Lagrangian function.

It is essential to achieve a reduction in the barrier function $B(x,r)$ at each iteration, because the barrier function serves as a convenient "merit function" for measuring progress toward x^*. The derivation of the barrier trajectory algorithm indicates that the search direction given by the solution of QP2 may not always be a descent direction for $B(x,r)$. Therefore, the null-space component of the search direction may alternatively be computed to minimize a quadratic approximation to the Lagrangian function, independent of the component in the range of the columns of $\hat{A}$. This alternative formulation of the search direction is necessary because of the quite different roles of the penalty and barrier parameters as the solution is approached.

2.2.2. Calculation of the search direction

At the beginning of the k-th iteration of the barrier trajectory algorithm, the same vectors and matrices are available as for the penalty trajectory algorithm. The iterates generated by the barrier trajectory algorithm necessarily lie in the strict interior of the feasible region; this algorithm is intended for use on problems where some or all of the problem functions may be ill-defined or undefined outside the feasible region.

The computational procedures followed during the k-th iteration are:

(1) Determine the set of "active" constraints, denoted by $\hat{c}$ (see Section 3.2); form the matrix $\hat{A}$, whose columns are the columns of $A^{(k)}$ corresponding to the active set. By construction, $\hat{A}$ has $\leq n$ columns.

(2) Factorize $\hat{A}$ such that

$$Q\hat{A} = \begin{bmatrix} R \\ --- \\ 0 \end{bmatrix} \ ,$$

$$Q^T Q = I \ ,$$

as before.

(3) Determine the Lagrange multiplier estimate λ by solving:

$$R\lambda = Y^T g^{(k)} \ ,$$

so that λ is the least-squares solution of $\min \| \hat{A}\lambda - g^{(k)} \|_2^2$; a similar procedure to that given for the penalty trajectory algorithm is followed if $\hat{A}$ is rank-deficient. If one or more components of λ are negative, the constraint corresponding to the most negative is deleted from the active set; the modified $\hat{A}$ is then factorized, and the new λ vector calculated for the redefined active set. Since the new $\hat{A}$ is simply the previous $\hat{A}$ with one column deleted, the new factorization can be obtained by a simple updating scheme (Gill, Golub, Murray, and Saunders, 1974).

(4) Determine the barrier parameter, r (Murray and Wright, 1976b).

(5) Determine the vector d according to the following rules. Define $s = \|\hat{c}\| + \|Z^T g^{(k)}\|$, and set $\varepsilon = \gamma\, r/s$, where $\gamma > 1$ (say, 2). Let $\hat{r}$ be the barrier parameter from the previous iteration; then:

$$\text{if } \lambda_i > \varepsilon, \text{ set } d_i = -\hat{c}_i + \frac{r}{\lambda_i} \ ;$$

$$\text{if } \lambda_i \leq -\varepsilon, \text{ set } d_i = -(1 - \frac{r}{\hat{r}})\ \hat{c}_i ;$$

$$\text{otherwise, } \qquad d_i = -\varepsilon \ .$$

(6) Compute the vector p_R, which is the solution of the linear system:

$$\hat{A}^T Y p_R = R^T p_R = d \ .$$

In this way, p_R satisfies the desired linear equality constraints of QP2 for those problem constraints for which λ_i is sufficiently positive; an alternative relationship is satisfied for each "active" constraint that has an insufficiently positive multiplier estimate. Again, the rank-deficient case is treated as for the penalty trajectory algorithm.

(7) Compute the modified Cholesky factorization of $Z^T S^{(k)} Z$ (as in the penalty trajectory algorithm); the factorization will be denoted by LDL^T.

(8) Determine $\bar{p}_N$ by solving:

$$LDL^T \bar{p}_N = -Z^T g^{(k)} .$$

Test whether:

$$\|\bar{p}_N\| \le M\|p_R\| \quad \text{and} \quad \|p_R\| \le M\|\bar{p}_N\| , \qquad (7)$$

for M a reasonably large positive number.

(a) If the test (7) is satisfied, obtain p_N by solving

$$LDL^T p_N = -Z^T(g^{(k)} + S^{(k)} Yp_R) ,$$

and define the trial search direction as:

$$p = Yp_R + Zp_N .$$

If p is not a descent direction for $B(x,r)$, re-define p as:

$$p = Yp_R + Z\bar{p}_N .$$

This latter definition is guaranteed to yield a descent direction for $B(x,r)$.

(b) If the test (7) is not satisfied, then adjust the scaling, as in the penalty trajectory algorithm.

(9) Determine a step length, α, that accomplishes a suitable reduction in $B(x,r)$, using special procedures designed for one-dimensional minimization with respect to the logarithmic barrier function (see Section 3.5.2). During the search procedure, record whether violation of any constraint currently considered "inactive" restricts the step length; if so, this constraint will be added to the active set at the next iteration.

(11) Set $x^{(k+1)}$ to $x^{(k)} + \alpha p$, and return to Step (1).

3. Some Considerations of Numerical Analysis in Implementation of the Trajectory Algorithms

In this section, we consider some selected aspects of implementation of the trajectory methods, from the viewpoints of numerical analysis and algorithm definition. It will be stressed throughout this discussion that an implementation could not achieve practical success if the definition of the algorithm depended critically on properties that hold only in a close neighborhood of x^*; the algorithm should produce sensible results, even when such conditions are not satisfied at the current point.

3.1. Selection of the Active Set for the Penalty Trajectory Algorithm

The "active set" of constraints is defined at each iteration of the penalty trajectory algorithm as the set of constraints whose values are less than a specified small positive number. With this definition, the active set is equivalent to the "violated set", and can easily be determined. Such a strategy is reasonable because the penalty trajectory algorithm is based on properties of the quadratic penalty function, and for a sufficiently large value of ρ, the set of constraints violated at $x_P^*(\rho)$ is identical to the set of constraints active at x^* (Fiacco and McCormick, 1968). After the first few iterations, the active set typically remains fixed for the rest of the computation.

It was noted in the definition of the algorithm that a special procedure is used when more than n constraints are violated at the beginning of an iteration. If more than n constraints are violated, and $\hat{A}$ has full rank, the search direction, p, is chosen to attempt to minimize $\|\hat{c}(x+p)\|^2$, by computing p as the solution of $\min\|\hat{c} + \hat{A}^T p\|^2$. The search direction in this case is calculated as follows:

(1) Factorize $\hat{A}^T$ in the form

$$Q\hat{A}^T = \begin{bmatrix} R \\ --- \\ 0 \end{bmatrix} , \qquad Q^T Q = I ,$$

where R is upper triangular and non-singular.

(2) Solve $Rp = -Y^T\hat{c}$, where the columns of Y are given by the first n rows of Q.

The computational procedure is very similar to the calculation of the Lagrange multiplier estimates, and relies on orthogonal transformations to reduce $\hat{A}^T$ to upper triangular form.

Normally, the condition that more than n constraints are violated occurs because the current point is a poor estimate of the solution, and does not hold at the next iterate. However, it is conceivable that this condition could exist even at x^*, so that possibly every iteration might be special. In this case, the Hessian matrix of the penalty function is not ill-conditioned as the penalty parameter approaches its limit. This special procedure has the same effect as choosing $\rho = \infty$ in the usual definition of the algorithm, and hence is equivalent to the Gauss-Newton method.

3.2. Selection of the Active Set for the Barrier Trajectory Algorithm

The criteria for selecting the active set in the barrier trajectory algorithm are not so straightforward as in the penalty trajectory algorithm. Because the barrier trajectory algorithm is a feasible-point method, all problem constraints are satisfied at every iteration, and the subset of active constraints must be determined by analysis of the behavior of the constraints as the computation proceeds.

The procedure for determining the initial active set is described in Murray and Wright (1976b), and has been satisfactory on the examples tested. The active set tends to be altered only during the early iterations, because of the possibility of misleading local indications that certain constraints are active. The following rules are used to modify the active set at each iteration:

(1) The constraint corresponding to the most negative Lagrange multiplier estimate (if one exists) is deleted, and the remaining multipliers are modified accordingly.

(2) If any constraint considered active appears to be bounded away from zero as the solution is approached, it is deleted from the active set. This decision is highly dependent on scaling; further discussion is given in Murray and Wright (1976b).

(3) If the step-length algorithm was restricted because a supposedly inactive constraint was violated, this constraint is added to the active set at the beginning of the next iteration, and will not be deleted during that iteration, regardless of the sign of its multiplier estimate.

(4) If the number of active constraints exceeds n following the addition of (3), the constraint with the largest value of $c_i(x)$ is deleted from the active set (a further scaling-dependent decision).

3.3. Use of Orthogonal Factorizations

A factorization involving reduction of $\hat{A}$ to triangular form by application of orthogonal transformations is used in several steps of the trajectory algorithms. Such a factorization is convenient and reliable for computation, and has many advantages over alternative procedures. For example, in some algorithms for solving P1, the matrix $\hat{A}^T\hat{A}$ is formed and used to solve linear systems; the poor numerical properties of this strategy are well-known (see Peters and Wilkinson, 1970), especially the possible squaring of the condition number that may result from the formation of $\hat{A}^T\hat{A}$. Furthermore, if the matrix $\hat{A}$ does not have full rank, $\hat{A}^T\hat{A}$ is singular, and some steps of the algorithm may then be undefined.

Computation of the complete orthogonal factorization of $\hat{A}$ means that steps of the trajectory algorithm can be defined (with relatively little extra work) even if $\hat{A}$ is rank-deficient (see Sections 3.3.1 and 3.3.2). Although only the singular value decomposition can fully reveal the closeness of the columns of $\hat{A}$ to linear dependence, the complete orthogonal factorization provides adequate information in many applications (see Golub, Klema, and Stewart, 1976), since the conditioning of the triangular matrix R serves to indicate the "conditioning" of $\hat{A}$.

3.3.1. Calculation of a Lagrange multiplier estimate

The Lagrange multiplier estimate at each iteration of the trajectory algorithms is computed as a least-squares solution of $\min\|\hat{A}\lambda-g\|_2^2$; this first-order estimate is acceptable, since the local rate of convergence of the trajectory methods is not restricted to the rate of convergence of the multipliers (see Wright, 1976). The orthogonal factorization of $\hat{A}$ can be used to calculate a minimum-length least-squares solution, even when $\hat{A}$ does not have full column rank; this alternative is not possible with techniques that involve forming $\hat{A}^T\hat{A}$.

When reducing $\hat{A}$ to upper triangular form, column interchanges are carried out so that the reduced column of largest magnitude is selected as the next column to be reduced; the matrix is considered to be numerically rank-deficient if the norms of all unreduced columns are less than a prescribed tolerance. In this way, all diagonal elements of R are bounded below by the specified tolerance. Although the ill-conditioning of R does not necessarily reveal itself by the presence of a diagonal element that is very small relative to the largest diagonal element, prevention of a too-small diagonal element is sufficient in many cases to control serious ill-conditioning of R.

3.3.2. Calculation of the search direction

The search direction in both trajectory algorithms is computed in two orthogonal components -- one in the range of the columns of $\hat{A}$, the other in the corresponding null space. This definition results from the characterization that the search direction must satisfy a set of linear equality constraints of the form:

$$\hat{A}^T p = b \ , \tag{8}$$

where b is some vector depending on the algorithm. If $\hat{A}$ has full rank, these equality constraints uniquely determine the component of p in the range of the columns of $\hat{A}$, which is calculated as follows.

The orthogonal matrix Q that reduces $\hat{A}$ to upper triangular form is explicitly formed, by multiplying out the orthogonal transformations used in the reduction. Once Q is available, its rows, appropriately partitioned, provide the required orthogonal bases for the range and null space of the columns of $\hat{A}$.

In the full rank case, it is straightforward to compute the component of p in the range of $\hat{A}$. Since $p = Yp_R + Zp_N$, the equality constraints (8) imply:

$$\hat{A}^T p = \hat{A}^T(Yp_R + Zp_N) = \hat{A}^T Yp_R = R^T p_R = b \ ,$$

which gives p_R as the solution of a non-singular triangular system.

If $\hat{A}$ is rank-deficient, the component p_R may be obtained as the least-squares solution of $\min \|\hat{A}^T p - b\|^2$, again using the complete orthogonal factorization of $\hat{A}$.

In either case, the calculation of p_R is completely straightforward.

3.4. Approximation of the Hessian of the Lagrangian Function

Alternative techniques for approximating the Hessian of the Lagrangian function under various circumstances will not be discussed in any detail (see Murray and Wright, 1976b, for such a discussion), but we shall consider one key property of the Hessian approximation.

The assumed second-order Kuhn-Tucker conditions imply that the matrix Z^TWZ must be positive definite at x^*, where Z is defined in terms of $\hat{A}(x^*)$, and W is the Hessian of the Lagrangian function; however, W itself need not be positive definite, or even non-singular, at x^* or in any neighborhood of x^*.

Certain approaches to solving P1 impose additional conditions on related matrices -- for example, methods involving augmented Lagrangian functions (see Powell, 1969; Fletcher, 1974) require that the penalty parameter be large enough so that the Hessian of the augmented function is positive definite. In both trajectory algorithms, however, only the projected Hessian, Z^TSZ, must be positive definite at every iteration, in order for the solutions of the quadratic programs QP1 and QP2 to be bounded. The vector p_N is the solution of a linear system:

$$Z^TSZ \ p_N = Z^Td \ , \tag{9}$$

for some vector d, and should be the step to the minimum of a quadratic function related to the Lagrangian function.

Accordingly, the matrix used to calculate p_N is always maintained as numerically positive definite. When Z^TSZ is updated by a quasi-Newton technique, positive definiteness is maintained by updating the Cholesky factorization of the projected matrix, as in revised quasi-Newton methods for unconstrained minimization (Gill and Murray, 1972b). When Z^TSZ is obtained from exact, or finite-difference approximations to, second derivatives, the modified Cholesky factorization of Z^TSZ is computed in order to solve the system (9). In all cases, the matrix used to solve (9) for p_N is represented as LDL^T, where L is unit lower traingular, and D is a diagonal matrix with all elements strictly positive. Such a procedure assures that the portion of the search direction in the null space of the columns of $\hat{A}$ is always well-defined, and bounded.

3.5. Step-Length Algorithms

3.5.1. Detection and correction of unbounded decrease of penalty function

Even when the problem P1 has a bounded solution, the corresponding penalty function or augmented Lagrangian function may be unbounded below, for arbitrarily large values of the penalty parameter (Powell, 1972). Accordingly, when executing a one-dimensional minimization with respect to a penalty function or augmented Lagrangian function, care must be exercised to avoid the possibility of taking an excessively large step.

In particular, the safeguarded cubic or quadratic step-length algorithms (Gill and Murray, 1974) used in the penalty trajectory algorithm require specification of an upper bound on the step length. In the current implementation of the penalty trajectory algorithm, the upper bound is set to correspond to a step of "reasonable" size, rather than an extremely large value. In some cases, the upper bound may impose an unnecessary limit on the stepsize; however, in general such a restriction will cause no serious loss of efficiency for the overall computation, since the next iteration usually corrects the possible poor scaling of the search direction. This conservative strategy is considered to be justified by the extreme difficulties that result if an enormous step is taken because the penalty function is unbounded below: either the next iterate is completely unreasonable, or a large number of evaluations of the problem functions are required before the unboundedness is detected.

In the penalty trajectory algorithm, it is considered that the penalty function may be unbounded along the given direction if the step taken is the specified upper bound. Almost always, the indicated unboundedness can be eliminated simply by increasing the penalty parameter.

3.5.2. Special techniques for the barrier trajectory algorithm

At each iteration of the barrier trajectory algorithm, a step-length algorithm is executed with respect to the logarithmic barrier function, which thus serves as a "merit function". Several authors (Fletcher and McCann, 1969; Lasdon, et al., 1973) have noted the deficiencies of standard step-length algorithms, which are usually based on approximation by low-order polynomials, when applied to the logarithmic barrier function. Therefore, the step-length algorithm of the barrier trajectory method makes use of special techniques that exploit the known properties of the logarithmic barrier function to allow more efficient estimation of an appropriate step length; these techniques are based on simple approximating functions that contain a logarithmic singularity. Only a small amount of additional calculation is required to fit the special approximating functions, and their use leads to a significant increase in efficiency of the one-dimensional minimization, compared to standard procedures (Murray and Wright, 1976a).

4. Conclusions

The penalty and barrier trajectory algorithms are based on the mathematical properties of the approach to x^* of the successive iterates generated by the quadratic penalty function and logarithmic barrier function, respectively. In theory, these algorithms have several desirable properties -- for example, their derivation does not depend on conditions that hold only in a close neighborhood of x^*, and their rate of convergence in the limit is arbitrarily close to that of linearly constrained Lagrangian algorithms (described in Robinson, 1972; Rosen and Kreuser, 1972). In practice, the current implementations of the trajectory algorithms have been successful on many problems, deliberately including examples for which the ideal assumptions are violated. The results thus far indicate that these algorithms compare favorably with similarly careful implementations of other algorithms to solve P1 (see Wright, 1976, for some typical numerical results).

The overall aim of this paper has been to illustrate some of the considerations of numerical analysis that enter the choice of computational procedures for selected aspects of the trajectory algorithms. Numerical analysis may not play a significant role in the process of verifying that the expected behavior of an algorithm under ideal conditions is displayed numerically. However, it is an elementary fact of numerical analysis that theoretically equivalent mathematical procedures do not yield equivalent, or even close, numerical results, and it is an elementary fact of life that hoped-for conditions are not always satisfied. Considerations of numerical analysis should, therefore, be applied to every aspect of the definition and implementation of optimization algorithms in general.

Acknowledgement

We thank Michael Saunders for his many helpful comments.

Research of this paper was partially supported by the Office of Naval Research under Contract N00014-75-C-0865; the Energy Research and Development Administration E(04-3)-326 PA #18; and the National Science Foundation Grant DCR75-04544. at Stanford University.

References

Fiacco, A.V. and McCormick, G.P. (1968). Nonlinear Programming: Sequential Unconstrained Minimization Techniques, John Wiley and Sons, New York.

Fletcher, R. (1974). "Methods Related to Lagrangian Functions," in Numerical Methods for Constrained Optimization (P.E. Gill and W. Murray, eds.), pp. 219-240, Academic Press, London and New York.

Fletcher, R. and McCann, A.P. (1969). "Acceleration Techniques for Nonlinear Programming," in Optimization (R. Fletcher, ed.), pp. 203-213, Academic Press, London and New York.

Gill, P.E., Golub, G.H., Murray, W., and Saunders, M.A. (1974). Methods for Modifying Matrix Factorizations, Mathematics of Computation, vol. 28, pp. 505-535.

Gill, P.E. and Murray, W. (1972a). Two Methods for the Solution of Linearly Constrained and Unconstrained Optimization Problems, Report NAC-25, National Physical Laboratory.

Gill, P.E. and Murray, W. (1972b). Quasi-Newton Methods for Unconstrained Optimization, Journal Inst. Math. Appl., vol. 9, pp. 91-108.

Gill, P.E. and Murray, W. (1974). Safeguarded Steplength Algorithms for Optimization using Descent Methods, Report NAC-37, National Physical Laboratory.

Golub, G.H., Klema, V., and Stewart, G.W. (1976). Rank Degeneracy and Least Squares Problems, Report CS-76-559, Stanford University.

Lasdon, L.S., Fox, R.L., and Ratner, M.W. (1973). An Efficient One-Dimensional Search Procedure for Barrier Functions, Mathematical Programming, vol. 4, pp. 279-295.

Murray, W. (1969a). Constrained Optimization, Report MA79, National Physical Laboratory.

Murray, W. (1969b). "An Algorithm for Constrained Minimization," in Optimization (R. Fletcher, ed.), pp. 247-258, Academic Press, London and New York.

Murray, W. and Wright, M.H. (1976a). Efficient Linear Search Algorithms for the Logarithmic Barrier Function, Report SOL 76-18, Systems Optimization Laboratory, Stanford University.

Murray, W. and Wright, M.H. (1976b). Trajectory Algorithms for Nonlinearly Constrained Optimization, in preparation.

Peters, G. and Wilkinson, J.H. (1970). The Least-Squares Problem and Pseudo-Inverses, Computer Journal, vol. 13, pp. 309-316.

Powell, M.J.D. (1969). "A Method for Nonlinear Constraints in Minimization Problems," in Optimization (R. Fletcher, ed.), pp. 283-298, Academic Press, London and New York.

Powell, M.J.D. (1972). "Problems Related to Unconstrained Optimization," in Numerical Methods for Unconstrained Optimization (W. Murray, ed.), pp. 29-55, Academic Press, London and New York.

Robinson, S.M. (1972). A Quadratically Convergent Algorithm for General Nonlinear Programming Problems, Mathematical Programming, vol. 3, pp. 145-156.

Rosen, J.B. and Kreuser, J. (1972), "A Gradient Projection Algorithm for Nonlinear Constraints," in Numerical Methods for Non-linear Optimization (F.A. Lootsma, ed.), pp. 297-300, Academic Press, London and New York.

Ryan, D.M. (1974). "Penalty and Barrier Functions," in Numerical Methods for Constrained Optimization (P.E. Gill and W. Murray, eds.), pp. 175-190, Academic Press, London and New York.

Wright, M.H. (1976). Numerical Methods for Nonlinearly Constrained Optimization, Report 193, Stanford Linear Accelerator Center, Stanford, California.

OPTIMIZATION ALGORITHMS DERIVED FROM NONQUADRATIC MODELS

by

J.S.KOWALIK
Department of Computer Science
and Mathematics
Washington State University
Pullman, Washington 99163

Abstract

The problem considered is the calculation of the least value of a general differentiable function of several variables. A brief review of two types of minimization methods based on nonquadratic models is offered. The first involves solving systems of linear equations in every iteration. The second is derived from a generalization of the standard conjugate gradient methods.

I. Introduction

The problem to be discussed is the computation of the unconstrained minimum value of a general differentiable multivariable function $f(\underset{\sim}{x})$. Most of the currently available algorithms use as a fundamental model the quadratic function

$$q(\underset{\sim}{x}) = \tfrac{1}{2}(\underset{\sim}{x}-\underset{\sim}{\hat{x}})^T Q(\underset{\sim}{x}-\underset{\sim}{\hat{x}}) \tag{1}$$

where Q is a positive definite matrix and $\underset{\sim}{\hat{x}}$ is the location of the minimum solution of (1). It is of interest to investigate more general models which may better reflect the local behavior of general continuous functions. It would be, for instance, desirable to develop methods that can successfully handle cases where the Hessian matrices are positive semi-definite at the solution or elsewhere.

An interesting departure from model (1) can be accomplished if we assume that

$$f(\underset{\sim}{x}) = F(q(\underset{\sim}{x})) \tag{2}$$

where F is a differentiable, strictly increasing function of a single variable $q>0$. The gradient of $f(\underset{\sim}{x})$ is

$$\underset{\sim}{g}(\underset{\sim}{x}) = F'Q(\underset{\sim}{x}-\underset{\sim}{\hat{x}}) \tag{3}$$

where

$$F' = \frac{dF}{dq} \tag{4}$$

and since $F'>0$ the minimum of $f(\underset{\sim}{x})$ takes place at $\underset{\sim}{x}=\underset{\sim}{\hat{x}}$. Furthermore, $f(\underset{\sim}{x})$ can be minimized in at most n steps if we use:

(a) a set of search directions mutually conjugate with regard to Q, i.e., $\underset{\sim}{d}_0, \underset{\sim}{d}_1, \ldots, \underset{\sim}{d}_{n-1}$ satisfying $\underset{\sim}{d}_i^T Q \underset{\sim}{d}_j = 0, i \neq j$, and,

(b) the optimal steps λ_i satisfying the equations

$$\underset{\sim}{x}_{i+1} = \underset{\sim}{x}_i + \lambda_i \underset{\sim}{d}_i \tag{5}$$

$$\underset{\sim}{d}_i^T \underset{\sim}{g}_{i+1} = 0 \tag{6}$$

for $i = 0,1,2,3,\ldots,n-1$

This property of $F(q(\underset{\sim}{x}))$ becomes obvious if we make the observation that the nonlinear function $F(q(\underset{\sim}{x}))$ leaves the isocontour curves of $q(\underset{\sim}{x})$ unchanged, altering only the function values on these curves.

Thus, the function $f(\underset{\sim}{x}) = F(q(\underset{\sim}{x}))$ can be minimized in a finite number of steps if we generate Q-conjugate search directions in the process of minimizing $f(\underset{\sim}{x})$. Obviously, a standard conjugate-gradient algorithm such as Fletcher-Reeves' or Davidon-Powell-Fletcher's will not be finite (unless $F(q) = q$) since the gradient vectors of $F(q(\underset{\sim}{x}))$ are multiples of the gradients of $q(\underset{\sim}{x})$ and this would alter the search directions generated by a standard conjugate gradient algorithm. However, a relatively simple modification of these algorithms will provide this property.

A pioneering work in this direction has been done by Fried [1971] who has constructed a modified Fletcher-Reeves method for minimizing $f(\underset{\sim}{x})$ of the form

$$f(\underset{\sim}{x}) = \frac{1}{p}q^p + \hat{f}, \quad p \geq 1 \tag{7}$$

where $\hat{f}$ is a constant value,

Function (7) satisfies the relationship

$$f(\underset{\sim}{x}) = \frac{1}{2p}(\underset{\sim}{x}-\underset{\sim}{\hat{x}})^T \underset{\sim}{g}(\underset{\sim}{x}) + \hat{f} \tag{8}$$

and has also been studied as an optimization model by Jacobson-Oksman [1972] and Kowalik-Ramakrishnan [1976]. They have used equation (8) as a basis for an optimization method which does not produce conjugate directions but is finite for (7). This method is briefly described in Section 2. A similar class of functions has been also studied by Davison and Wong [1974] . Section 3 shows a derivation of modified conjugate gradient methods minimizing $f(x) = F(q(x))$ and a detailed analysis of $f(x) = \frac{1}{p}q^P$. Section 4 contains numerical results and Section 5 is a conclusion.

2. A method based on the homogeneous model

We can rewrite (8) in the form

$$f(x) = \frac{1}{\gamma}(x-\hat{x})^T g(x)+\hat{f} \qquad (9)$$

where $\gamma = 2$ for function (1).
If we define

$$a^T = (\hat{x}^T,\gamma,\tilde{f}) \qquad (10)$$

$$\tilde{f} = \gamma\hat{f} \qquad (11)$$

$$y^T = (g(x)^T, f(x), -1) \qquad (12)$$

$$v = x^T g(x) \qquad (13)$$

then equation (9) can be restated as:

$$y^T a = v \qquad (14)$$

If we compute y and v at $n+2$ distinct points (n is the size of the x vector) we get a system of linear equations

$$Ya = w \qquad (15)$$

where

$$Y = \begin{pmatrix} y_1^T \\ y_2^T \\ \cdot \\ \cdot \\ \cdot \\ y_{n+2}^T \end{pmatrix} \qquad w = \begin{pmatrix} v_1 \\ v_2 \\ \cdot \\ \cdot \\ \cdot \\ v_{n+2} \end{pmatrix}$$

and if Y is nonsingular we can solve (15) for a which contains the solution to the minimization problem $\min_x f(x)$.

Clearly, it is possible to get an algorithm minimizing functions satisfying (8) in $n+2$ steps. For general functions an iterative procedure can be constructed where a sequence of approximations $x_0, x_1, \ldots, x_i, \ldots$ is generated such $x_i \to x^*$ where x^* is a local minimum. In this procedure, equation (15) becomes iterative $Y_i a_i = w_i$ and Y_i is obtained by removing one row from Y_{i-1} and inserting the new row $y^T(x_i)$ computed at x_i. The vector w_i is also modified by one component. The method requires solving systems of equations which differ by one row. Solutions to such systems can be obtained inexpensively and accurately by using different factorization methods (Kowalik-Ramakrishnan [1976] and Kowalik-Kumar [1976]).

3. A modified conjugate-gradient method.

In the method of conjugate gradients of Fletcher and Reeves [1974] the search direction is calculated as a sum of two vectors,

$$d_i = -g_i+\beta_i d_{i-1}, \; i\geq 1 \qquad (17)$$

and

$$d_0 = -g_0 \qquad (18)$$

The scalar multiplier β_i can be determined from the conjugacy condition

$$d_i^T y_{i-1} = 0 \qquad (19)$$

where

$$y_{i-1} = g_i - g_{i-1} = Q(x_i - x_{i-1}) \qquad (20)$$

Equations (19) and (20) give

$$\beta_i = \frac{g_i^T y_{i-1}}{d_{i-1}^T y_{i-1}} \qquad (21)$$

and from (17)

$$d_i = -g_i + \frac{g_i^T y_{i-1}}{d_{i-1}^T y_{i-1}} \; d_{i-1} \qquad (22)$$

Equation (22) is still suitable in the case when $f(x) = F(q(x))$ except that we have to redefine y_{i-1}. It can be seen from (22) that if we define y_{i-1} as

$$y_{i-1} = g_i/F'_i - g_{i-1}/F'_{i-1} \qquad (23)$$

then the search directions generated by (22) will be collinear for any $F(q(x))$. In other words, if (23) is used in equation (22), then $d_0, d_1, \ldots, d_i, \ldots, d_{n-1}$ are scaled conjugate gradient directions produced by the standard Fletcher-Reeves method for $F(q(x)) = q(x)$.

Now

$$\beta_i = \frac{g_i^T(\rho_i g_i - g_{i-1})}{d_{i-1}^T(\rho_i g_i - g_{i-1})} \qquad (24)$$

where

$$\rho_i = \frac{F'_{i-1}}{F'_i} \qquad (25)$$

Equation (24) can also be written as:

$$\beta_i = \frac{||\underset{\sim}{g}_i||^2}{||\underset{\sim}{g}_{i-1}||^2}\ \rho_i \qquad (26)$$

The modified formulas (24) and (26) to compute β_i can be useful if we know how to compute ρ_i. Clearly ρ_i depends on the choice of F(q). Since available numerical results suggest that the function $\frac{1}{p}q(x)^p$ may be a good model for unconstrained optimization we will assume now that

$$f(\underset{\sim}{x}) = F(q(\underset{\sim}{x})) = \frac{1}{p}q^p, p>o \qquad (27)$$

The gradient of $f(\underset{\sim}{x}_i)$ is

$$g_i = F'_iQ(\underset{\sim}{x}_i-\hat{\underset{\sim}{x}}) \qquad (28)$$

and

$$F'_i = q_i^{p-1} = (pf_i)^{\frac{p-1}{p}}$$

$$= (pf_i)^{1-t} \qquad (29)$$

where

$$t = \frac{1}{p} \qquad (30)$$

Furthermore

$$\rho_i = \frac{F'_{i-1}}{F'_i} = \left(\frac{f_i}{f_{i-1}}\right)^{t-1} \qquad (31)$$

and we assume that t is an unknown value that has to be determined at each step of the iterative process. From (28) we have

$$\frac{F'_{i-1}}{F'_i} = \frac{\underset{\sim}{g}^T_{i-1}(\underset{\sim}{x}_i-\hat{\underset{\sim}{x}})}{\underset{\sim}{g}^T_i(\underset{\sim}{x}_{i-1}-\hat{\underset{\sim}{x}})}$$

$$= \frac{\underset{\sim}{g}^T_{i-1}\underset{\sim}{x}_i-\underset{\sim}{g}^T_{i-1}\hat{\underset{\sim}{x}}}{\underset{\sim}{g}^T_i\underset{\sim}{x}_{i-1}-\underset{\sim}{g}^T_i\hat{\underset{\sim}{x}}} \qquad (32)$$

and

$$\frac{F'_{i-1}}{F'_i} = \frac{\underset{\sim}{g}^T_{i-1}\underset{\sim}{x}_i-\underset{\sim}{g}^T_{i-1}\underset{\sim}{x}_{i-1}+2pf_{i-1}}{\underset{\sim}{g}^T_i\underset{\sim}{x}_{i-1}-\underset{\sim}{g}^T_i\underset{\sim}{x}_i+2pf_i} \qquad (33)$$

since

$$\underset{\sim}{g}^T_i\hat{\underset{\sim}{x}} = \underset{\sim}{g}^T_i\underset{\sim}{x}_i-2pf_i \qquad (34)$$

But

$$\underset{\sim}{g}^T_i\underset{\sim}{x}_{i-1}-\underset{\sim}{g}^T_i\underset{\sim}{x}_i = \lambda_{i-1}\underset{\sim}{g}^T_i\underset{\sim}{d}_{i-1} = 0 \qquad (35)$$

and from (31) and (33)

$$\left(\frac{f_i}{f_{i-1}}\right)^{t-1} = \frac{\lambda_{i-1}\underset{\sim}{d}^T_{i-1}\underset{\sim}{g}_{i-1}+2pf_{i-1}}{2pf_i} \qquad (36)$$

or

$$\left(\frac{f_i}{f_{i-1}}\right)^{t} = t\frac{\lambda_{i-1}\underset{\sim}{d}^T_{i-1}\underset{\sim}{g}_{i-1}}{2f_{i-1}}+1 \qquad (37)$$

Introducing an abbreviated notation for the coefficients of (37) we get

$$a_i^T = tb_i+1 \qquad (38)$$

Equation (38) (first time obtained by Fried [1971]) is solved for a nontrivial unique root t_i which in turn is used to calculate ρ_i from equation (31).

As indicated by Goldfarb [1972] the Quasi-Newton methods can also be modified to minimize f(x) = F(q(x)) in a finite number of steps. The effect of nonlinearly scaling the objective function on the Quasi-Newton methods has been also investigated by Spedicato [1976].

Computational results

The extended Fletcher-Reeves method (EFR) using formulas (26) and (31), where t is calculated by solving (38), has been tested on several standard functions and compared with the classical Fletcher-Reeves method (FR). An identical one-dimensional search routine has been used in both methods. The programs have been written in FORTRAN and computations performed in double precision on IBM/360.

The following problems have been tried.

Test problem 1 (Quartic with singular Hessian)

$$f(\underset{\sim}{x}) = (x_1+10x_2)^2+5(x_3-x_4)^2+(x_2-2x_3)^4+10(x_1-x_4)^4$$

The starting point is (3,-1,0,1) and the function has a minimum of zero at (0,0,0,0).

Test Problem 2 (homogeneous quartic)

$$f(\underset{\sim}{x}) = [\tfrac{1}{2}\underset{\sim}{x}^TQx+\underset{\sim}{b}^T\underset{\sim}{x}+0.25]^2$$

$$Q = \begin{bmatrix} 4.5 & 7 & 3.5 & 3 \\ 7 & 14 & 9 & 8 \\ 3.5 & 9 & 8.5 & 5 \\ 3 & 8 & 5 & 7 \end{bmatrix}, \qquad \underset{\sim}{b} = \begin{bmatrix} -0.5 \\ -\ 1 \\ -1.5 \\ 0 \end{bmatrix}$$

The starting point is (4,4,4,4) and the

function has a minimum of zero at (.5,-.5,.5,0).

Test problem 3 (Rosenbrock's function)

$$f(\underset{\sim}{x}) = 100(x_i^2-x_2)^2+(1-x_1)^2$$

The starting point is (-1.2,1) and the function has a minimum of zero at (1,1).

Test problem 4 (a Hilbert quadratic form)

$$f(\underset{\sim}{x}) = \tfrac{1}{2}\underset{\sim}{x}^T H\underset{\sim}{x}^{\,k}, \quad k=2,3$$

where

$$h_{ij} = \frac{1}{i+j-i}, \quad i,j=1,2,\ldots,5$$

The starting point is (-3,-3,...,-3). The solution is at (0,0,...,0).

Test problem 5 (4-dimensional Rosenbrock's function)

$$f(\underset{\sim}{x}) = 100(x_1^2-x_2)^2+(1-x_1)^2+90(x_3^2-x_4)^2$$
$$+(1-x_3)^2+10.1[(x_2-1)^2+(x_4-1)^2]$$
$$+19.8(x_2-1)(x_4-1)$$

The function has a minimum of zero at (1,1,1,1). Table I shows the experimental results for test Problem I which is a quartic function whose Hessian is singular (has two zero eigenvalues) at the solution $x^T=(0,0,0,0)$. It looks as in such cases $f(\underset{\sim}{x}) = \frac{1}{p}q^p$ is a better optimization model. This may be due to the fact that $f(\underset{\sim}{x}) = \frac{1}{p}q^p$ has a singular Hessian at the solution for $p>1$.

As expected, the EFR method converges after approximately n steps for the test problems 2 and 4 (Table III, rows 3 to 6). It should be pointed out that in order to achieve convergence for these functions in approximately n steps, the EFR method had to be implemented with an accurate one-dimensional search. More interesting are tests involving general functions, such as ROSENBROCK and 4-DIMENSIONAL ROSENBROCK. In these tests we have also implemented the EFR method with an accurate one-dimensional search. Our objective has been to find if EFR could generate better search directions; that is, converge in fewer iterations with the optimal step size strategy.

Tables III and IV, and row 3 of Table II show sample runs. We have had, however, a couple of cases where EFR has been only marginally better or slightly outperformed by the FR method.

In most test cases the value of ρ_i differed significantly from 1 in the initial stage of optimization and approached 1 in the final phase of convergence. For example, the run shown in Table III ρ_i varied from 138 to .572 between the second and the twelfth iteration and was quite close to 1 afterwards. This was not the case with Problem 1 where even at the very end of the convergence process ρ_i assumed values as large as 15.

Conclusion

The preliminary results presented in this paper suggest that the EFR method based on the model $f(\underset{\sim}{x})=\frac{1}{p}q(\underset{\sim}{x})^p, p>0$ may deal better with functions whose Hessian is singular at some points along the optimization path or at the solution.

There is no strong indication that MFR based on this model can perform significantly better that FR when optimizing general nonlinear continuously differentiable functions.

Another open problem is the influence of the one-dimensional search accuracy on the performance of the EFR method.

And the final comment. It would be interesting to design other methods based on the model $f(\underset{\sim}{x})=F(q)$ where $F'(q)>0$ for $q>0$ and see of they offer any advantage over $f(\underset{\sim}{x})=q$.

TABLE I

Iteration Number	FR	EFR
0	$.2150 \times 10^{3}$	$.2150 \times 10^{3}$
10	$.4793 \times 10^{-1}$	$.1403 \times 10^{-1}$
20	$.1715 \times 10^{-4}$	$.9883 \times 10^{-6}$
30	$.1882 \times 10^{-5}$	$.5172 \times 10^{-10}$
50	$.7786 \times 10^{-8}$	----

Quartic with singular Hessian

TABLE II

Test Function	FR	EFR	Solution Accuracy
Homogeneous Quartic	29*,203**	5*,43**	10^{-11}
Rosenbrock	46*,100**	38*,86**	10^{-8}
Hilbert k=2	20*	4*	10^{-11}
Hilbert k=3	25*	4*	10^{-14}
$f(x) = q^{.5}_{***}$	64*	7*	10^{-7}
$f(x) = q^{.1}_{***}$	52*	6*	.69476

*Number of iterations

**Number of function and gradient evaluations

***q is a quadratic function defined in problem 2

TABLE III

Iteration Number	FR	EFR
0	$.19192*10^{5}$	$.19192*10^{5}$
12	$.1482*10^{0}$	$.7080*10^{-2}$
20	$.3491*10^{-2}$	$.44196*10^{-4}$
25	$.1009*10^{-3}$	$.1509*10^{-5}$
30	$.7474*10^{-4}$	$.9292*10^{-11}$

4-DIMENSIONAL ROSENBROCK

$x_0^T = (-3,-1,-3,-1)$

TABLE IV

Iteration Number	FR	EFR
o	$.4166*10^{2}$	$.4166*10^{2}$
20	$.8421*10^{1}$	$.7821*10^{0}$
60	$.1624*10^{0}$	$.5787*10^{-1}$
80	$.2568*10^{-1}$	$.33923*10^{-6}$

4-DIMENSIONAL ROSENBROCK

$x_0^T = (-1.2,1,1.2,1)$

Acknowledgement

The author would like to thank Mrs. Swarn Kumar and Mr. Emmanuel Kamgnia for their help in running numerical experiments.

References

1. Davison, E.J. and P. Wong, A Robust Conjugate-gradient algorithm which minimizes L-functions, Control Systems Report No. 7313, University of Toronto, Dept. of Engineering, Toronto, 1974.
2. Fletcher, R., and C.M. Reeves, Function Minimization by Conjugate Gradients, Comput. J, Vol.7, 1964.
3. Fried, I., N-step Conjugate Gradient Minimization Scheme for Nonquadratic Functions, AIAA Journal, Vol.9, No.11, 1971.
4. Goldfarb, D., Variable Metric and Conjugate Direction Methods in Unconstrained Optimization: Recent Developments, ACM Proceedings, 1972.
5. Spedicato, E., A Variable-Metric Method for Function Minimization Derived from Invariancy to Nonlinear Scaling, Journal of Optimization Theory and Applications, Vol.20, No.3, Nov. 1976.
6. Jacobson, D.H. and Oksman, W., An Algorithm that Minimizes Homogeneous Function of N Variables in N+2 Iterations and Rapidly Minimizes General Functions, Journal of Mathematical Analysis and Applications, ol.38, 1972.
7. Kowalik, J.S. and S.P. Kumar, Fast Givens Transformations Applied to the Homogeneous Method, IFIP Congress 77, submitted.
8. Kowalik, J.S. and Ramakrishnan, K.G., A Numerically Stable Optimization Method Based on a Homogeneous Function, Mathematical Programming, Vol.11, 1976.

ALGORITHMS FOR A CLASS OF "CONVEX" NONLINEAR INTEGER PROGRAMS*

R. R. Meyer and M. L. Smith

Mathematics Research Center and Computer Sciences Department,
University of Wisconsin-Madison

Abstract

Algorithms are given for the efficient solution of the class of nonlinear integer programs with separable convex objectives and totally unimodular constraints. Because of the special structure of this problem class, the integrality constraints on the variables can be easily handled. In fact, the integrality constraints actually make the problem "easier" than its continuous version, for in the case that bounds are available on the problem variables, the first of the proposed algorithms yields the optimal solution by the solution of a *single*, easily-generated linear program. For the cases in which bounds are not available for the variables or the sum of the variable ranges is very large, other algorithms are discussed that yield the solution after a finite number of linear programs and require less storage than the first algorithm.

1. Introduction

Consider the nonlinear integer program†

$$\text{(1.1)}\qquad \begin{aligned} &\min_{x} && \sum_{i=1}^{n} f_i(x_i) \\ &\text{s.t.} && Ax = b,\ x \geq 0,\ x \text{ integer} \end{aligned}$$

where $x = (x_1,\ldots,x_n)^T$, each f_i is *convex* on a *closed interval* $B_i \subset [0,+\infty)$ and A is *totally unimodular* $m \times n$ matrix. The intervals B_i, which are assumed to be given, are assumed to *cover* the feasible set of (1.1) in the sense that $\Omega \equiv \{x \mid Ax = b, x \geq 0, x \text{ integer}\} \subseteq \{x \mid x_i \in B_i;\ i=1,\ldots,n\}$. (No differentiability or continuity properties are needed or assumed for the f_i, but convexity of f_i on B_i implies continuity on the interior of B_i.)

Problems of the class (1.1) arise in logistic and personnel assignment applications, and have been the subject of a number of studies [1], [4], [6], [7], [8] that deal with the special case in which $Ax = b$ consists of the single constraint $\sum_{i=1}^{n} x_i = K$ (see also [5] for an algorithm for this special case). Dantzig [3, p. 498] considers the case in which the constraints $Ax = b$ correspond to supply-demand constraints in a bi-partite network.

The problem class (1.1) has the remarkable property (shown in [5]) that the integrality constraints actually make the problem easier to solve than its corresponding continuous version.

Specifically, it was shown in [5], that if there exist *known* non-negative integers ℓ_i, u_i such that $B_i = [\ell_i, u_i]$ for $i = 1,\ldots,n$ (i.e., there exist known bounds for the feasible set of (1.1)), then (1.1) may be solved by solving the single *linear program* constructed by (1) replacing each f_i by an appropriate piecewise linear "approximation" and (2) deleting the integrality constraints. It was also shown in [5] that, if the intervals B_i are *infinite* (or if the sum of the ranges of the x_i is very large), an algorithm based on the solution of a *finite* number of similarly constructed linear programs is guaranteed to yield the optimal solution within a finite number of iterations, under the assumption that the given problem of class (1.1) has an optimal solution. In this report we consider in more depth the question of the trade-offs between number of LP's solved, storage required, and number of function evaluations for four specific algorithms of the general types described previously. A numerical example is discussed in Section 4.

---

† When modified in the obvious fashion, the algorithm to be described below can be used to solve more general problems in which (1) the constraints $Ax = b$ are replaced by a system of equations and inequalities with a totally unimodular constraint matrix, and (2) integer upper and lower bounds on individual variables are added. Alternatively, such constraints can be converted into the form (1.1) through the addition of integer slack variables, and the resulting coefficient matrix will be totally unimodular.

* Supported in part by the National Science Foundation under Contract No. DCR74-20584 and in part by the United States Army under Contract No. DAAG29-75-C-0024.

2. Basic Algorithms

In order to describe in a compact manner algorithms for the problem (1.1) we will introduce some notation to represent certain piecewise-linear approximations to (1.1). Letting R_i $(i=1,\ldots,n)$ be finite sets of non-negative integers, we define a linear programming approximation to (1.1) as the problem $P(R_1,\ldots,R_n)$ given by:

$$
\begin{aligned}
&\min_{x,\lambda} \sum_{i=1}^{n} \sum_{j \in R_i} f_i(j)\lambda_{i,j} \\
&\text{s.t.}\quad Ax = b, \quad x \geq 0 \\
(2.1)\qquad &\sum_{j \in R_i} j\lambda_{i,j} = x_i, \quad \sum_{j \in R_i} \lambda_{i,j} = 1 \ (i=1,\ldots,n) \\
&\lambda_{i,j} \geq 0 \quad \forall\, i,j.
\end{aligned}
$$

The problem $P(R_1,\ldots,R_n)$ can be thought of as a separable programming approximation to (1.1) in which the integrality constraints are deleted and the breakpoint-sets are given by the sets R_i $(i=1,\ldots,n)$. The significance of this LP approximation, as shown in [5], is that (1) an extreme point of (2.1) will have x_i integer-valued for $i=1,\ldots,n$, and (2) if x_i^* $(i=1,\ldots,n)$ is part of an optimal solution (x^*,λ^*) of (2.1) and the breakpoint-sets R_i have the property that (for $i=1,\ldots,n$)

$$(2.2)\qquad [\{x_i^*-1, x_i^*, x_i^*+1\} \cap B_i] \subseteq R_i ,$$

then an optimal solution to (1.1) is obtained by setting $x_i = x_i^*$ $(i=1,\ldots,n)$. (It should be noted that it is essential to employ a "separable programming approximation" to (1.1), since other piecewise-linear approximations that agree with f_i at the integer points in $B_i (i=1,\ldots,n)$ may not have the required extreme point integrality property - see [5].)

We will now consider three algorithms for (1.1) corresponding to three different procedures for constructing the R_i.

Algorithm 1 (Single LP, maximal R_i)

This algorithm is applicable only if there are known bounds ℓ_i, u_i (which will, without loss of generality, be assumed to be non-negative integers) such that $B_i = [\ell_i, u_i]$, i.e. if x is feasible for (1.1), then $\ell_i \leq x_i \leq u_i$ for $i=1,\ldots,n$. It has the advantage that it yields a solution to (1.1) via the solution of a single easily-constructed linear program. Specifically, let each R_i consist of the integers in the interval $[\ell_i, u_i]$, and solve the LP (2.1), then note that the optimality condition (2.2) is satisfied by x^* if (x^*,y^*) is an optimal extreme point of the LP (if (2.1) is infeasible, then so is (1.1)). ■

Algorithm 1 provides a very efficient, one-step approach to the solution of (1.1) as long as the problem $P([\ell_1,u_1],\ldots,[\ell_n,u_n])$, which will have $2n + \sum_{i=1}^{n}(u_i-\ell_i)$ variables and $m + 2n$ constraints, does not exceed the capacity of the available linear programming algorithm. In this regard, it should be noted that many commerical LP "packages" have separable programming and/or generalized upper bound capabilities that take advantage of the special structure of (2.1). It is also possible to modify the data handling in the simplex algorithm so that a distinct column is not needed for each $\lambda_{i,j}$, since, for a given i, the columns corresponding to the $\lambda_{i,j}$ can differ only in two entries. Furthermore, if the constraints $Ax = b$ can be given a network interpretation, efficient algorithms for network optimization can be applied rather than the ordinary simplex algorithm (see, for example, [2]).

However, if known bounds ℓ_i, u_i are not available or if the attempted implementation of Algorithm 1 would lead to storage problems, then alternative approaches are possible because the optimality conditions (2.2) do not require a "full grid" of points. The next two algorithms take advantage of this fact. To get starting "grids" for the next two algorithms, two cases should be considered: (1) if bounds ℓ_i, u_i are known, then the initial breakpoint-sets $R_i^{(0)}$ needed for the algorithms can be taken as any integer sets containing at least ℓ_i and u_i, and (2) if finite bounds are not available for all variables, let $\bar{x}$ be an extreme point of $\{x \mid Ax=b, x \geq 0\}$ (such an $\bar{x}$ may be determined by the simplex method, and it will be integer), and set $R_i^{(0)}$ to $\{\ell_i, \bar{x}_i, u_i\}$, where ℓ_i and u_i $(i=1,\ldots,n)$ are estimates (which need not be rigorous) for lower and upper bounds on an optimal solution of (1.1). (If a good guess is available for the optimal solution of (1.1), the corresponding breakpoints should also be included in the $R_i^{(0)}$.) In both cases, the initial LP considered will be feasible if and only if (1.1) is feasible, so we will assume in the algorithms below that feasibility has been established.

Algorithm 2 (Multiple LP's, intermediate R_i's)

At the kth iteration $(k=0,1,2,\ldots)$, we assume that a feasible solution $x^{(k)}$ to (1.1) has been obtained by solving the problem $P(R_1^{(k)},\ldots,R_n^{(k)})$. If the optimality conditions (2.2) are satisfied by $x^* = x^{(k)}$ and $R_i = R_i^{(k)}$, then terminate with $x^{(k)}$ as optimal solution of (1.1). Otherwise obtain new breakpoint-sets $R_i^{(k+1)}$ by adding to each $R_i^{(k)}$ the points of $[\{x_i^{(k)}-1, x_i^{(k)}, x_i^{(k)}+1]\} \cap B_i] \setminus R_i^{(k)}$, (these are the points "missing" from the optimality conditions) and let the (k+1)st iterate $x^{(k+1)}$ be the value of x in an optimal extreme point of $P(R_1^{(k+1)},\ldots,R_n^{(k+1)})$ ($x^{(k+1)}$ will be integer).

We also assume that the algorithm tests $x^{(k)}$ for optimality in $P(R_1^{(k+1)},\ldots,R_n^{(k+1)})$ and terminates by declaring $x^{(k)}$ optimal for (1.1) if this test is satisfied.

(Since $(x^{(k)}, \lambda^{(k)})$ would normally be used as a starting basic feasible solution for $P(R_1^{(k+1)}, \ldots, R_n^{(k+1)})$, this is a natural assumption.) ▪

Algorithm 3 (Multiple LP's, minimal R_i's)

Proceed as in Algorithm 2, except that if $x^{(k)}$ and the $R_i^{(k)}$ do not satisfy the optimality conditions of (2.2), then the breakpoint-sets $R_i^{(k+1)}$ for the next iteration are taken to be the sets $\{x_i^{(k)}-1, x_i^{(k)}, x_i^{(k)}+1\} \cap B_i \ (i=1, \ldots, n)$. ▪

Algorithms 2 and 3, which represent opposite ends of the column-generation spectrum, will converge to an optimal solution of (1.1) in a finite number of iterations, provided that (1.1) actually has an optimal solution. This result follows from a finiteness theorem [5] for integer programs with separable convex objective functions. These algorithms also in general avoid the problem generation and storage problems that may occur in some cases in Algorithm 1. In certain instances, however, difficulties with storage limitations and/or speed of convergence might also arise with Algorithms 2 and 3, and some additional computational refinements are described in the following section.

3. Some Computationally Useful Modifications

Although the algorithms of the previous section are guaranteed to display either one-step or finite convergence under rather weak hypotheses, theoretical finite convergence of course does not necessarily imply that an optimal solution will be obtained within the time or storage available to solve the problem.

If Algorithm 1 can be employed without exceeding the capacity of the available LP or network code, then time and storage will not be problems unless $\sum_{i=1}^{n} (u_i - \ell_i)$ turns out very large. On the other hand, Algorithm 2, starts out with relatively small breakpoint-sets, but there is no control on the size of these sets, and thus no guarantee that problem size limits might not be exceeded.

Although Algorithm 3 employs the minimal breakpoint-sets needed to establish optimality, one would not expect it to be very efficient, since the value of a variable can change by at most one unit at each iteration, and since much of the computed information on values of the f_i may be discarded at each iteration. In addition, slow convergence might also occur in Algorithm 2 in the case that lower and upper bounds are not known and at least one of the estimates ℓ_i, u_i is consistently violated by the iterates, since Algorithm 2 allows only a single unit change beyond the estimated bounds at each iteration.

Algorithm 4, to be described below (see also Figure 1) avoids these potential problems, and also makes use of the possibility of a lower bound on the optimal solution of (1.1). To set up the linear program for determining a lower bound, let $x^{(k)}$ denote the optimal solution of the most recent LP solved, and let R_i' be a set such that, for each i, either $x_i^{(k)} \in R_i'$ or $x_i^{(k)}-1 \in R_i'$, and such that $j \in R_i'$ implies $(j+1) \in B_i$ (while any set of breakpoints with these properties will be suitable for lower bound generation, larger R_i' will yield larger LP's and, in general, better lower bounds).

A lower bound on the optimal value of (1.1) may then be obtained by solving the following LP, since the objective function of the LP is no greater than the objective function of (1.1) on the feasible set Ω:

$$\begin{aligned}
&\min_{x,y,\delta} \sum_{i=1}^{n} y_i \\
&\text{s.t.} \quad Ax = b, \ x \geq 0 \\
(3.1) \qquad & \qquad y_i \geq f_i(j) + \delta_{i,j}(f_i(j+1) - f_i(j)) \\
& \qquad x_i = j + \delta_{i,j} \\
& \qquad (j \in R_i', \ i = 1, \ldots, n) .
\end{aligned}$$

(Of course, the lower bound generated by solving (3.1) may turn out to be $-\infty$, in which case it would not be useful. In many cases, however, additional information such as non-negativity can be included in the lower-bounding LP in order to prevent unboundedness. For example, if the functions $f_i(x_i)$ are known to be non-negative for non-negative x_i (in many applications the f_i are exponentials or posynomials), then the additional constraints $y_i \geq 0$, when added to the constraints of (3.1) will prevent unboundedness of the objective function.) Since Algorithms 2 and 3 (and Algorithm 4 below) generate feasible solutions to (1.1), it is possible to use a lower bound on the optimal value of (1.1) in a termination criterion if one sets an optimality tolerance on the gap between the lower bound and the value of the best feasible solution. (In general, the feasible solution with the best objective value will be the last iterate, but an exception to this might occur if an optimal solution $(\underline{x}, \underline{y}, \underline{\delta})$ to (3.1) had the property that $\underline{x}$ was integer. In general, a solution of (3.1) will not have this integrality property, but if it does, then $\underline{x}$ will be a feasible solution of (1.1), and may have an objective function value better than that of the last iterate, in which case this iterate should be replaced by $\underline{x}$. Note that if $\underline{x}$ is integer and $\sum_{i=1}^{n} f_i(\underline{x})$ coincides with the optimal value of (3.1), then $\underline{x}$ solves (1.1). See Figure 1 for the details of how these possibilities may be taken into account.)

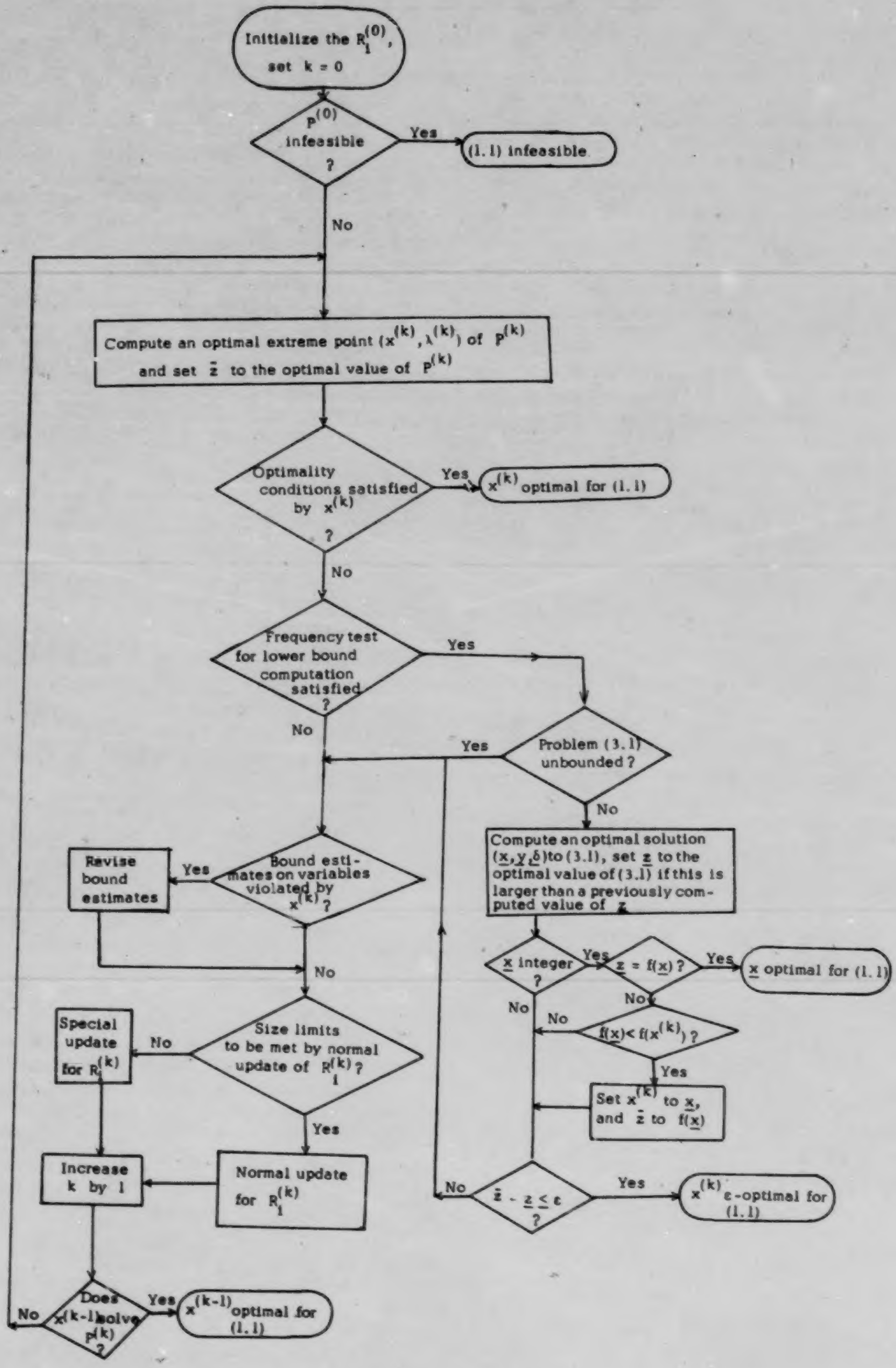

Initialize the $R_i^{(0)}$, set $k = 0$
$P^{(0)}$ infeasible ?
Yes
(1.1) infeasible
No
Compute an optimal extreme point $(x^{(k)}, \lambda^{(k)})$ of $P^{(k)}$ and set $\bar{z}$ to the optimal value of $P^{(k)}$
Optimality conditions satisfied by $x^{(k)}$?
Yes
$x^{(k)}$ optimal for (1.1)
No
Frequency test for lower bound computation satisfied ?
Yes
No
Problem (3.1) unbounded ?
Yes
No
Compute an optimal solution $(\underline{x}, \underline{y}, \underline{\delta})$ to (3.1), set $\underline{z}$ to the optimal value of (3.1) if this is larger than a previously computed value of $\underline{z}$
$\underline{x}$ integer ?
Yes
$\underline{z} = f(\underline{x})$?
Yes
$\underline{x}$ optimal for (1.1)
No
No
No
$f(\underline{x}) < f(x^{(k)})$?
Yes
Set $x^{(k)}$ to $\underline{x}$, and $\bar{z}$ to $f(\underline{x})$
$\bar{z} - \underline{z} \le \varepsilon$?
No
Yes
$x^{(k)}$ ε-optimal for (1.1)
Bound estimates on variables violated by $x^{(k)}$?
Yes
Revise bound estimates
No
Size limits to be met by normal update of $R_i^{(k)}$?
No
Special update for $R_i^{(k)}$
Yes
Normal update for $R_i^{(k)}$
Increase k by 1
Does $x^{(k-1)}$ solve $P^{(k)}$?
Yes
$x^{(k-1)}$ optimal for (1.1)
No

Figure 1. Flowchart for Algorithm 4

The statement of Algorithm 4 assumes that the following parameters have been supplied: (a) a bound on the total number of $\lambda_{i,j}$ variables that will be allowed in any LP, (b) increments α_i, β_i to be used for revising lower and upper bound estimates (these are not used if rigorous lower and upper bounds are known), (c) an optimality tolerance ε and a parameter establishing the frequency of the lower bound computations (this could be based on clock time or number of iterations).

Algorithm 4 (Multiple LP's, bounded storage, optimality tolerance)

Proceed as in Algorithm 3, except that:

(a) when the update of the $R_i^{(k)}$ would violate the bound on the number of λ_{ij}, remove from the breakpoint sets $R_i^{(k)}$, prior to the update, all indices other than those corresponding to the current values of the upper and lower bound estimates;

(b) when an iterate violates the current value ℓ_i' of the lower bound estimate, ℓ_i' is updated to $\max\{0,\ell_i'-\alpha_i\}$, and when an iterate violates the current value u_i' of the upper bound estimate, u_i' is updated to $u_i'+\beta_i$;

(c) a lower bound is periodically computed by solving a problem of the form (3.1), and the algorithm is terminated if the objective function value of the best feasible solution obtained thus far lies within ε of the lower bound (if the t^{th} problem of the form (3.1) has an optimal value $\underline{z}^t > -\infty$, the constraint $\sum_{i=1}^{n} y_i \geq \underline{z}^t$ may be added to the (t+1)st problem of the form (3.1) in order to guarantee monotonicity of the lower bounds; note that the dual simplex algorithm may be used with the optimal solution from the t^{th} problem serving as the initial solution of the dual of the (t+1)st problem). ■

For notational convenience in the flowchart for Algorithm 4, $\sum_{i=1}^{n} f_i(x_i)$ is denoted by $f(x)$ and $P(R_1^{(s)},\ldots,R_n^{(s)})$ is denoted by $P^{(s)}$.

4. Numerical Example

In this section we present a numerical example to illustrate the algorithmic ideas introduced in the previous sections.

The problem dealt with has the form

$$\min \sum_{i=1}^{15} w_i(1-q_i)^{x_i}$$

$$\text{(4.1)} \quad \text{s.t.} \sum_{i=1}^{10} x_i = b_1, \quad \sum_{i=6}^{15} x_i = b_2$$

$$0 \leq x \leq u, \quad x \text{ integer}$$

where the data and optimal solution, x^* are given in Table 1. Using the "complete grid" approach of Algorithm 1, the resulting LP has 17 equations and 262 variables, and the use of a small, locally-written LP package yielded the optimal solution x^* of Table 1.

By contrast, a column-generation procedure of the type described in Algorithm 2 started with 60 variables[†] and terminated with an optimal solution (x^* of Table 1) in 7 interations, the final LP solved having 131 variables.

i	w_i	q_i	u_i	x_i^*
1	9.2	0.31	16	12
2	1.0	0.45	16	4
3	7.6	0.23	19	14
4	0.6	0.09	10	2
5	8.8	0.15	10	10
6	4.2	0.21	11	11
7	3.2	0.15	17	12
8	3.4	0.01	20	0
9	8.8	0.79	16	3
10	6.6	0.41	15	7
11	1.2	0.71	17	3
12	4.6	0.77	12	4
13	0.8	0.79	13	2
14	3.0	0.21	20	13
15	1.2	0.07	20	12
$b_1 = 75$, $b_2 = 67$, optimal value $\cong 7.7586$				

Table 1. Data for Numerical Example

References

1. L. B. Boza, "The Interactive Flow Simulator: A System for Studying Personnel Flows," paper presented at the ORSA/TIMS Joint National Meeting, San Juan, Puerto Rico, October 1974.
2. A. Charnes, Fred Glover, David Karney, D. Klingman, Joel Stutz, "Past, Present and Future of Large Scale Transshipment Computer Codes and Applications," Research Report CS 131, Center for Cybernetic Studies, The University of Texas, Austin, October 1974.
3. George B. Dantzig, Linear Programming and Extensions, Princeton University Press, Princeton, 1963.
4. O. Gross, "Class of Discrete Type Minimization Problems," RM-1644, Rand Corp., Santa Monica, 1956.

[†] In addition to ℓ_i and u_i, $R_i^{(0)}$ included the greatest integer in $(\ell_i+u_i)/2$.

5. R. R. Meyer, "A Class of Nonlinear Integer Programs Solvable by a Single Linear Program," Computer Sciences Technical Report #267, University of Wisconsin-Madison, February 1976.

6. Thomas L. Saaty, Optimization in Integers and Related Extremal Problems, McGraw-Hill, New York, 1970

7. R. E. Schwartz and C. L. Dym, "An Integer Maximization Problem," Opns. Res. 19 (1971), 548-550.

8. Iram J. Weinstein and Oliver S. Yu, Comment on an Integer Maximization Problem," Opns. Res. 21 (1973), 648-650.

EXTREME POINT RANKING ALGORITHMS:
A COMPUTATIONAL SURVEY

Patrick G. McKeown
The University of Georgia

ABSTRACT

Since it has been long known that the optimal solution to the minimization of a concave objective function over a convex set will occur at an extreme point of the convex set, one method of solving this type of problem is to rank these extreme points. In the case where the objective function in nonlinear and the constraint set is linear, there have been numerous articles, more conceptual than computational, on the application of an extreme point ranking algorithm as a solution procedure. In this paper, we will review the usefulness of this general type of procedure to various problems by attempting to combine the available computational literature with computational results of the author that have not been previously presented.

I. Introduction

This paper will be concerned with problems of the following general type:

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } x \in S \qquad\qquad (P) \end{aligned}$$

where $S = \{x \mid Ax=b, x \geq 0\}$, and $f(x)$ is concave. A is assumed to be $m \times n$, b is $m \times 1$, and x is $n \times 1$.

It was shown by Hirsch and Hoffman [12] that an optimal solution to P would occur at an extreme point or vertex of S. Hence, to find an optimal solution to P it is "only" necessary to search the vertices of P until an optimal solution is found and proved. If f(x) is linear, i.e.,

$$f(x) = \sum_{j=1}^{n} C_i x_i \quad ,$$

then the well-known simplex method is a very efficient procedure for carrying out this search. However, if f(x) is nonlinear, say, quadratic or integer, there does not exist a simplex type of algorithm. It is the nonlinear case that we shall be concerned with here.

Since no "direct" optimization techniques exist for the case where f(x) is nonlinear, we shall look at two approaches to searching the extreme points of S. First, one might wish to use a linear under approximation of f(x), say L(x), such that $L(x) \leq f(x)$ $x \in S$. In this case, to show that x^* is an optimal solution to P, we need only rank the vertices of S until the vertex x° is found such that $L(x^\circ) \geq f(x^*)$. At this point, all vertices that could possibly be optimal have been ranked. This is proved by Cabot and Francix [3].

The second case applies to the case where f(x) is a sum of a linear and a nonlinear portion, i.e.,

$$f(x) = \sum_{j=1}^{n} C_i x_i, + g(x)$$

where $g(x) \geq 0$ for $x \geq 0$.

In this situation, one may seek to find a lower bound on g(x), say G. Then we may use the linear portion of f(x) plus the lower bound as an under approximation of f(x). Obviously, for $x \geq 0$, $f(x) \geq C^T x + G$. Hence, if x^* is optimal for P, then we need only rank the vertices of S until a point x° is found such that $C^T x^\circ + G \geq f(x^*)$. This proves that x^* is optimal. The best example of this is the fixed charge problem.

In order to rank the extreme points of S, we need to use in both cases above a result also first proved by Murty [20] as Theorem 1 below:

Theorem 1: If $E_1, E_2, \ldots, E_K$ are the first K vertices of a linear under approximation problem which are ranked in nondecreasing order according to their objective value, then vertex E_{K+1} must be adjacent to one of $E_1, E_2, \ldots, E_K$.

Simply put, this says that vertex 2 will be adjacent to the optimal solution to the linear under approximation and vertex 3 will be adjacent to vertex 1 or vertex 2. This, then, gives us a procedure for ranking the vertices if all adjacent vertices can be found. It is this "if" that quite possibly has accounted for the few number of computational papers relative to the number of conceptual works. This comes about due to the possibility of degeneracy in S. If S is degenerate, then there may exist multiple bases for the same vertex. This implies that all such bases must be available before one can be sure that all adjacent vertices have been found. Finding all such bases for finding and "scanning" all adjacent vertices can be quite cumbersome. As we shall see later, a recent application of Chernikova's work [5] has been shown to be a way around the problem of degeneracy. This will be discussed in more detail when the fixed charge problem is explored.

The literature has been found to refer to basically four types of problems:

(i) fixed charge problems,

(ii) traveling salesman,
(iii) quadratic assignment, and
(iv) concave quadratic programming problems.

We shall briefly discuss the conceptual background of each problem and then present and discuss the available computational applications of the extreme point ranking procedure to each problem. Where appropriate, we will present previously unpublished work of the author on the problem at hand. Finally, we will attempt to draw conclusions about the efficiency of this type of procedure.

II. The Fixed Charge Problem

One of the first types of problems recognized as being of the form specified as P previously is the linear fixed charge problem. This problem is formulated as P_F below:

$$\begin{aligned} &\text{Min} \quad c^t x + f^t y \\ &\text{S.t.} \quad x \epsilon S \qquad\qquad (P_F) \\ &y_j = \begin{cases} 1 \text{ if } x_j > 0 \\ 0 \text{ if } x_j = 0 \end{cases} \end{aligned}$$

In this case, f and y are nx1 and all other dimensions are as before. The f values are the fixed or "set-up" costs while the c values are the continuous costs. Hirsch and Dantzig [10, 11] first formulated P_F and recognized that an optimal solution of P_F would occur at an extreme point of s if the f_j's are non-negative.

A special subset of the fixed charge problem is the fixed charge transportation problem (FCTP) where the A matrix takes on the form of the Hitchcock Transportation Problem constraint set, i.e.,

$$\sum_{i=1}^{M} x_{ij} = d_j \qquad j = 1,\dots,N$$

$$\sum_{j=1}^{N} x_{ij} = s_i \qquad i = 1,\dots,M$$

$$\text{and} \quad \sum_{i=1}^{M} s_i = \sum_{j=1}^{N} d_j .$$

Here, M = number of supply points and N = number of demand points. Balinski [2] showed that a linear under approximation of the objective function of the FCTP could be found by first letting $u_{ij} = \text{Min } \{s_i, d_j\}$ and then setting

$$L_T(x) = \sum_{i=1}^{M} \sum_{j=1}^{N} (C_{ij} + f_{ij}/u_{ij})\, x_{ij} .$$

Although Hirsch and Dantzig had formulated the fixed charge problem and Balinski had found approximate solutions earlier, Murty [20] first suggested the use of extreme point ranking algorithm for solving this class of problems. He showed that if a lower bound on the sum of fixed charges could be found, say F_0, then the optimal solution to P_F, (x^*,y^*), could be found by ranking the vertices of the corresponding continuous problem (all $f_j = 0$) until a point x_0 is found such that $C^T x + F_0 \geq C^T x^* + f^T y$. The values of (x^*,y^*) may be found by checking each extreme point to determine whether a lower value of $C^T x + f^T y$ existed. When the point x_0 is found, the solution (x^*,y^*) is optimal. Murty did not discuss any computational results and left several unresolved problems with the solution procedure.

The first unresolved problem was to find F_0. Murty suggested that the m smallest fixed charges be summed. This may be easily seen to be inadequate for problems with greater-than constraints or for degenerate problems. This method also does not attempt to find a lower bound that is feasible.

Secondly, Murty did not discuss an adequate method of handling degeneracy in finding adjacent extreme points other than determining all bases and applying the simplex method to each one in turn.

Although Murty did not present computational results of using his extreme point ranking method, he did hypothesize that it would probably be more efficient in those cases where the continuous portion $(C^T x)$ dominated the fixed portion $(f^T y)$. This was later shown by McKeown in his dissertation [16] and in a later article [17]. He effected an implementation of Murty's procedure by resolving the two problems mentioned earlier. He showed that a lower bound on the fixed charges, F_0, could be found by solving by linear programming the set-covering problem P_β below.

$$\begin{aligned} &\text{Min } F_0 = \sum_{j=1}^{n} f_j y_j \\ &\text{S.t.} \quad \sum_{j=1}^{n} \delta_{ij} y_j \geq \beta_j, \\ &\qquad\qquad y_j \geq 0, \qquad\qquad (P_\beta) \end{aligned}$$

where $\delta_{ij} = \begin{cases} 1 \text{ if } a_{ij} > 0 \\ 0 \text{ if } a_{ij} \leq 0 \end{cases}$

and $\beta_j = 1$ (except for the FCTP where β_j values greater than 1 were used to account for the number of cells in a row necessary to possibly achieve feasibility).

This method was found to yield better (larger) values of f_0 than that originally suggested by Murty while also handling the problems with greater-than constraints and degeneracy mentioned earlier.

To handle the problem caused by degeneracy of S in finding adjacent vertices, a modification of work by Chernikova [5] was used to determine adjacent vertices of a degenerate vertex [25].

Computational experience with this implementation of Murty's suggestion bore out the original conjecture that ease of solution would largely be a function of the relative size of the fixed and continuous portions of the objective function. Two types of problems were tested. First, a group of general linear fixed charge problems generated by Steinberg [26] were run using a FORTRAN code on an IBM 360/175. These were 5x10 problems with equal-to constraints which were randomly generated such that $0 \leq C_j \leq 20$ and $0 \leq d_j \leq 999$. Five of these problems were tested and solved regardless of relative costs.

In the second case, a set of nine fixed charge transportation problems originally generated by Gray [9] were tested. In this case, these problems ranged from M=3 and N=4 to M=6 and N=8. Three of these problems had a relatively large continuous portion while the remaining six were fixed charge dominant. In the former case, the procedure was quite efficient regardless of problem size while in the latter case, the algorithm proved to be inefficient in that none of the six could be solved due to storage overruns.

In the later article [17], McKeown expanded the computational experience by using more variations of Steinberg's problems, i.e., 10x20's and problems with greater-than constraints. In these problems, results were encouraging as long as the relative cost sizes were favorable. For the problems with equal-to constraints, the fixed costs dominated due to M structural variables always being basic. However, in the problems with greater-than constraints, the reverse was true and the results showed that extreme point ranking was much more efficient for this latter class of problems. He also tested for the effect of degeneracy on FCTP's by comparing degenerate versions of originally non-degenerate problems. The results appeared to show that degeneracy is not a problem as long as the relative costs are favorable.

In summary, Murty's extreme point ranking procedure does appear to be efficient in those cases where the fixed cost portion is small compared to the continuous portion.

Two articles have appeared that use cutting plane variations of Murty's procedure. Cabot [4] suggested the use of Tui [29] cuts. The original Tui algorithm involved determining a local minimum, say $\bar{x}$, over s. Then a hyperplane is passed through the convex polytope in such a way that all extreme points of s with value greater than the local minimum, $\bar{x}$, are excluded. These are combined with the linear under approximation for fixed charge transportation problems,

$$L_T(x) = \sum_{j=1}^{N} \sum_{i=1}^{N} (C_{ij} + f_{ij}/u_{ij})\, x_{ij} .$$

He used two problems generated by Gray for his testing. Both problems had M=4 and N=6. Problem 1 was continuous cost dominant while Problem 2 was fixed cost dominant. Before using the Tui cuts, Cabot attempted to solve both problems via extreme point ranking using $L_T(x)$. As would be expected, he quite easily solved Problem 1 but was unable to prove optimality for Problem 2 after ranking over 400 extreme points. He then used a combined Tui Cut - extreme point ranking procedure to solve both problems with equal efficiency. This insensitivity to relative costs became more apparent when he devised 30 test problems by randomly generating new objective functions for Problems 1 and 2 where the ratio of fixed costs to continuous cost ranged from 5 to 200. For these problems his procedure appeared to be equally efficienct regardless of the ratio of fixed to continuous costs in that he solved 23 of the 30 test problems. He handled degeneracy by using a pertubation scheme, which while resolving degeneracy, introduced numerous additional extreme points.

In another paper, Taha [28] combined extreme point ranking with "Glover [8] cuts" to solve general linear fixed charge problems. He defines a Glover Cut to be one which separates a given extreme point from the convex polytope. He used as a linear under approximation the continuous portion of objective function, i.e.,

$$L_g(x) = \sum_{j=1}^{n} C_j x_j ,$$

and used these Glover Cuts to reduce the number of extreme points to be ranked. In addition, he used a technique suggested by Balas [1] of dropping constraints associated with the degenerate basic variable to redefine the polytope. It appears that this was done to insure that the Glover Cut was defined rather than to find adjacent extreme points. This procedure also tended to generate additional extreme points to be considered. Taha solved randomly generated problems of size as large as 15x20 with $0 \leq C_j \leq 800$ and $0 \leq F_j \leq 100$ in an average of 47 seconds on an IBM 360/50. As may be noted, these problems appear to be continuous cost dominated. This conjecture seems to be borne out by the second set of test problems where for $0 \leq F_j \leq 300$, we find that the solution times have gone up by a factor of four. Since these problems were randomly generated, it is highly unlikely that any were degenerate so we do not know the possible effect of degeneracy on the solution procedure.

One point that Taha makes is that the use of a linear under approximation for general problems similar to that suggested by Balinski [2], i.e.,

$$L_g(x) = \sum_{j=1}^{n} (C_j + F_j/u_j)\, x_j$$

$$\text{where } u_j \geq x_j \quad \forall_j ,$$

is made difficult by the need to solve a family of linear programming problems to find the u_m's.

Recently, the author investigated further the use of $L_T(x)$ as a linear under approximation for ranking extreme points for the fixed charge transportation problem as suggested by Cabot [4]. In this work, he used a ranking procedure specifically developed for transportation polytopes [19]. The nine problems developed by Gray [9] were used as bench marks for comparison with other procedures. In Table 1 below, we show the results from this computational testing. In this we have shown, for each problem the size (MxN), the relative sizes of the fixed and continuous portions of optimality (F*/C*), the number of extreme points ranked to prove optimality, the solution time on the UNIVAC 1110, the solution time for Kennington's [13] branch-and-bound procedure on the CDC Cyber 70, and Gray's original times on the Burroughs 5500.

As we can see, only on problems 1, 3, and 8 are our times competitive with those of Kennington or Gray. These problems are the ones with large continuous portions, and this is to be expected. This procedure does seem to be better than the original ranking algorithm suggested by Murty in that at least we were able to solve all of the problems, while McKeown could only solve 1, 3, and 8 using Murty's algorithm [20].

Table 2 below compares extreme point ranking results using the original Murty algorithm and the Balinski Approximation. Both of these were run by the author using the previously discussed ranking procedure and are for the same nine problems in

TABLE 1

COMPUTATIONAL RESULTS USING BALINSKI'S APPROXIMATION TO SOLVE FCTP'S

Problem	Size (MxN)	F*/C*	Number Ranked	1110 Time	5500 Time	Cyber 72 Time
1	3x4	59/270	3	0.09	7.7	.08
2	4x6	95/108	732	53.36	32.6	1.35
3	4x6	52/1947	2	0.15	26.3	.08
4	4x8	108/164	756	124.0	171.4	1.10
5	5x7	117/126	***	-----	263.8	4.93
6	5x7	173/143	154	18.70	149.9	0.99
7	5x7	1472/166	***	-----	97.0	1.48
8	5x7	59/2230	2	.20	3262.8	0.11
9	6x8	120/194	***	-----	1510.1	14.52

(All time in seconds)

*** - procedure aborted due to storage overflow

TABLE 2

COMPARISON OF MURTY'S APPROACH TO BALINSKI'S APPROXIMATION

		Number of Extreme Points Ranked	
Gray Problem	Size MxN	Murty	Balinski
1	3x4	29	3
2	4x6	***	732
3	4x6	2	2
4	4x8	***	756
5	5x7	***	***
6	5x7	***	154
7	5x7	***	***
8	5x7	9	2
9	6x8	***	***

*** - stopage overflow

In this we note that the Balinski Approximation is quite a bit more efficient than the Murty approach both in number of points ranked for those problems that were solved, as well as for number of problems actually solved.

III. Concave Quadratic Programming Problems

If we define

$$f_Q(x) = C^T x + x^T D_x \qquad (P_Q)$$
$$\text{s.t. } x \epsilon S$$

then we may say that the optimal solution to P_Q will occur at an extreme point of S under any one of the following conditions as enumerated by Cabot and Francis [3]:

1. The matrix D is negative definite or negative semidefinite.
2. Problem P_Q is a quadratic programming formulation of a problem occurring in bimatrix games.
3. Problem P_Q is a quadratic assignment problem.

These were not meant to be inclusive of all conditions under which the optimal solution of P_Q occurred at an extreme point, but rather, were examples of such conditions. We will consider condition 3 in a subsequent section, and there does not seem to have been any computational experience in solving bimatrix games via extreme point ranking. As a result of this, we will restrict our attention to problems fitting condition 1, i.e., D negative semi-definite or negative definite.

For this case, Cabot and Francis show that if we solve the family of linear problems, P_u^j, below for each column of D, d_j, then the minimum value of the objective function, u_j, can be used to determine a linear under approximation, L(x).

$$\text{Min } u_j = d_j^T x \qquad (P_u^j)$$
$$\text{S.t. } x\ S$$

Now, using u_j, we write P_{QL} as the linear under approximation problem as:

$$\text{Min } L(x) = \sum_{j=1}^{n} (C_j + u_j)\, x_j \qquad (P_{QL})$$
$$\text{S.t. } x \epsilon S.$$

A major drawback in using this approach is the necessity of computing the u_j values. One special case where this is not a problem is the quadratic transportation problem P_{QT} as formulated below:

$$\text{Min } \sum_{i=1}^{M} \sum_{j=1}^{N} (C_{ij}x_{ij} + d_{ij}x_{ij}^2)$$
$$\text{S.t. } \sum_{i=1}^{M} x_{ij} = A_j \qquad j=1,\ldots,N$$
$$\sum_{j=1}^{N} x_{ij} = B_i \qquad i=1,\ldots,M \qquad (P_{QT})$$
$$x_{ij} \geq 0 \text{ for all } i,j$$
$$\text{and } d_{ij} \leq 0.$$

This problem was formulated as a way of modeling the marginally decreasing cost characteristics of the private motor freight industry [24]. As Cabot and Francis note, $u_{pq} = d_{pq}[\min(B_i, A_j)]$ and we may easily formulate the objective function for (P_{QL}) for the quadratic transportation problem.

It should be noted that in the case of where one is attempting to model the decreasing marginal cost characteristics over various routes by use of P_{QT}, it would be expected that the quadratic portion will be small relative to the continuous portion. In fact, we would expect that $C_j x_j + d_{ij} x_{ij}^2$ would be positive non-decreasing over the entire range of values of x_{ij}, or $C_{ij} \geq |2d_{ij} u_{ij}|$ for each i,j. We would also expect that the effectiveness of (P_{QL}) for solving the problem by extreme point ranking would increase as C_{ij} increases relative to $d_{ij} U_{ij}$.

To test this, we used two sets of five randomly generated quadratic transportation problems ranging in size from 3x4 to 5x7. In each case we used the Chernikova algorithm as modified by McKeown and Rubin [19] to handle degenerate problems. The continuous cost ranges are shown below:

$$\text{Case I:} \quad 2\mu_{ij} d_{ij} < c_{ij} \leq 5000$$

$$\text{Case II:} \quad 2\mu_{ij} d_{ij} + 500 < c_{ij} \leq 9999$$

In all problem sets, the quadratic costs were between 0 and 10. In problem 5, these costs were between 0 and 20. The results of this testing are shown in Table 3.

From these results, we note some differences in run time as we move from Case I problems to Case II. This is felt to be due to the increasing dominance of continuous costs over quadratic costs in the latter case. We also note an increase in both parameters as we move to large problems for Case I. This is to be expected due to increasing problem size. This does not happen in Case II since very few solutions need be ranked in this case.

Problem Set 5 is the same as Problem Set 2 except that the range of quadratic costs has been increased. Here we note a great increase in solution parameter values in Set 5 over Set 2 for Case I. This is due to the decrease in dominance of continuous costs over quadratic costs.

TABLE 3

SOLUTION TIMES AND NUMBER OF EXTREME POINTS
RANKED FOR VARIOUS CONTINUOUS COST RANGES

Problem Set	Size (MxN)	d_{ij} Range	Case I Time	Case II Time
1	3x4	0-10	.09	.08
2	4x6	0-10	.34	.31
3	4x8	0-10	.86	.84
4	5x7	0-10	1.78	.85
5	4x6	0-20	1.13	.41

In terms of results, these problems tend to be much more promising than any of the others discussed in this paper. If further research demonstrates that, for problems of the nature discussed by Oi and Hurter, the costs are indeed marginally decreasing with dominant continuous costs, then this ranking procedure may be useful for solving such transportation problems.

IV. The Quadratic Assignment Problem

One area of mathematical programming where it also has long been known that the solution would occur at an extreme point is that of variations of the assignment problem. One such variation, the well known traveling salesman problem will be discussed in a later section. Here we will discuss the assignment problem where there are interactions between the assignments [7]. This is commonly referred to as the quadratic assignment problem and may be formulated as P_{AQ} below:

$$\begin{aligned} \text{Min} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} x_{ij} + \sum_{i}^{n} \sum_{p}^{n} \sum_{j}^{n} \sum_{q}^{n} K_{ijpq} x_{ij} x_{pq} \\ \text{S.t.} \quad & \sum_{i=1}^{n} x_{ik} = 1 \qquad k = 1,\dots,n, \\ & \sum_{K=1}^{n} x_{ik} = 1 \qquad i = 1,\dots,n, \qquad\qquad (P_{AQ}) \\ & x_{ik} \geq 0, \end{aligned}$$

where K_{ijpq} is the cost of an assignment of i to j and p to q.

That the solution of P_{AQ} would occur at an extreme point of the constraint set was first noted by Gilmore [7]. A linear under approximation to P_A was suggested by Lawler [14] and may be formulated as follows: denote for each i, j, a minor of K_{ijpq} as $K^{(ij)}$ and denote the value of the solution of the (n-1)x(n-1) assignment problem corresponding to each minor as $Z^{(ij)}$. We also define a cost $f_{ij} = Z^{(ij)} + C_{ij}$. The solution to the assignment problem for the matrix $F = \{f_{ij}\}$ is a lower bound on the solution to P_{QA}. As a result, we may use F, the assignment problem with cost coefficients $\{f_{ij}\}$, as a linear under approximation to P_{QA} for extreme point ranking purposes. This is an analogous procedure to that discussed for finding a linear approximation to quadratic programming problems with concave objective function discussed earlier [3].

In some problems, we may define

$$K_{ijpq} = t_{ip} d_{jp} .$$

This is true in the Koopmans-Beckman single commodity problem.

An extreme point ranking procedure for solving this and other non-convex quadratic minimization problems was suggested by Cabot and Francis [3] using the linear under approximation F. This procedure is essentially that discussed earlier for this type of linear under approximation in that extreme points of S are ranked until a value x_0 is found such that $L_A(x_0) \geq f(x^*)$ where x^* is the optimal solution.

Using this approach to the quadratic assignment problem, Fluharty [6] wrote a master's thesis on the problem. In that work, he tested various problems generated by previous researchers who had worked on the quadratic assignment problem. Using a procedure suggested by Murty [21] for determining adjacent extreme points for assignment problems, he tested problems for $4 \leq n \leq 12$ with mixed results. Only in those cases where the continuous values, i.e., $\{C_{ij}\}$, were large as compared to the quadratic values was he consistently assured of not having to enumerate all n! extreme points to reach a solution. For a problem with n = 12,

$0 \leq C_{ij} \leq 99$, and $0 \leq t_{ip},d_{iq} \leq 10$, the procedure overran available core storage. However, when the max C_{ij} was increased to 999, the problem was solved by ranking only 100 extreme points. In Table 4, we see the results of Fluharty's work.

TABLE 4

SUMMARY OF FLUHARTY'S RESULTS

Problem	Size(N)	C_{ij} Range	t_{ip},d_{iq} Range	Solutions Ranked
1	4	(0,0)	(0,20)	20
2	5	(0,0)	(0,5)	11
3	5	(1,5)	(0,5)	37
4	6	(0,0)	(0,10)	720
5	6	(0,9)	(0,10)	202
6	6	(0,99)	(0,10)	10
7	7	(0,99)	(0,9)	135
8	7	(0,0)	(0,10)	5040
9	7	(0,99)	(0,10)	170
10	7	(0,9)	(0,10)	1848
11	7	(0,0)	(0,99)	3201
12	7	(0,99)	(0,99)	3212
13	8	(0,0)	(0,10)	***
14	8	(0,9)	(0,10)	***
15	8	(0,99)	(0,10)	121
16	10	(0,99)	(0,9)	***
17	10	(0,999)	(0,9)	188
18	12	(0,0)	(0,10)	***
19	12	(0,99)	(0,10)	***
20	12	(0,999)	(0,10)	100

***-storage overflow

Once again the importance of relative size of linear vs. nonlinear portions of the objective function becomes evident for the use of extreme point procedures.

V. Traveling Salesman Problem

With all the work on using extreme point ranking algorithms to solve various problems with linear constraints, it might be surprising to find that very little work has been done on the traveling salesman problem. It can be easily seen that the optimal traveling salesman tour can be found by ranking the solutions to the assignment problem until the tour is found. However, our research has shown that only some very early work by Murty and Karel [23] has been documented. However, they opted to move onto their branch and bound algorithm [15] instead of continuing work on extreme point ranking methods.

In this early paper, a method for ranking the assignments was presented together with a procedure to avoid generating adjacent assignments that contained subtours. Using this procedure, a ten city randomly generated problem was solved "...by hand, and the time taken was about half an hour." Also, a 20 city symmetric problem was solved. This "...involved the solving of 10 different assignment problems of sizes ranging from 16 to 20. On the Burroughs 220 computer,..., this took about 10 minutes in all." [20] As mentioned earlier, this work, though not published, resulted in a branch and bound algorithm. Also, the method for ranking the assignments presented in this paper was later published [21].

One other bit of unpublished work in this area came to light from Sweeney and Williams [27]. It was stated by them that for a problem on the order of 40 cities, approximately 5000 assignments were ranked without a tour being found.

One further comment on the traveling salesman problem concerns another paper by Murty [22] on the tours of the traveling salesman problem. In that paper, he proves that all solutions that are tours can be found by determining the vertices adjacent to the diagonal assignment solution, i.e., $x_{ii} = 1$, $i = 1, \ldots, N$. However, due to the large number assignments adjacent to any other assignment, unless some efficient procedure can be found for generating these tours in a non-increasing costwise manner, this does not appear promising.

VI. Conclusions and Directions for Future Research

From our look at the use of extreme point ranking procedures to solve problems with concave objective functions and linear constraint sets, it would seem safe to say that the efficiency of this procedure is extremely dependent upon the objective function. In those problems where the objective function was approximately linear, the procedure could be expected to be very efficient. However, as the non-linearity of the objective function increases, the efficiency of the procedure decreases markedly.

Several areas of future research into the use of this procedure present themselves. One would be to combine the use of the Tui cut with methods for efficiently determining adjacent vertices in the presence of degeneracy. Another possibly fertile area of research would be to look more closely at the general linear fixed charge problem with greater-than constraints. Finally, research might be profitable in implementing Cabot and Francis' approach to more general concave quadratic programming problems.

REFERENCES

[1] Balas, E., "Intersection Cuts - A New Type of Cutting Plant for Integer Programming," Operations Research 19, 19-39 (1971).

[2] Balinski, M. L., "Fixed Cost Transportation Problems," Naval Research Logistics Quarterly, 8, No. 2, 41-54 (1961).

[3] Cabot, A. V. and R. L. Francis, "Solving Certain Nonconvex Quadratic Minimization Problems by Ranking the Extreme Points," Operations Research 18, 82-86 (1970).

[4] Cabot, A. V., "Variations on a Cutting Plane Method for Solving Concave Minimization Problems with Linear Constraints," Naval Research Logistics Quarterly, 21, No. 2, (1974).

[5] Chernikova, N. V., "Algorithm for Finding a General Formula for the Non-negative Solutions of a System of Linear Inequalities," U.S.S.R. Computational Mathematics and Mathe-

matical Physics.

[6] Fluharty, R., "Solving Quadratic Assignment Problems by Ranking the Assignments," Unpublished Master's Thesis, Ohio State University, 1970.

[7] Gilmore, P. C., "Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem," SIAM Journal 10, 305-313 (1962).

[8] Glover, F., "Convexity Cuts and Cut Search," Operations Research, 21, 123-134 (1973).

[9] Gray, P., "Exact Solution of the Fixed Charge Transportation Problem," Operations Research, 19, No. 6, 1529-1537 (1971).

[10] Hirsch, W. M. and Dantzig, G. B., "The Fixed Charge Problem," Rand Corporation Memo P. 648, The Rand Corporation, Santa Monica, California (1954).

[11] Hirsch, W. M. and Dantzig, G. B., "The Fixed Charge Problem," Naval Research Logistics Quarterly, 15, [illegible] 3, 413-424 (1968).

[12] Hirsch, W. M. and Hoffman, A. J., "Extreme Varieties, Concave Functions, and The Fixed Charge Problem," Communications on Pure and Applied Mathematics, 14, No. 3, 355-370 (1961).

[13] Kennington, J., "The Fixed Charge Transportation Problem: A Computational Study with a Branch-and-Bound Code," Technical Report CP 73033, Southern Methodist University, 1975.

[14] Lawler, E. L., "The Quadratic Assignment Problem," Management Science 9, 586-599 (1963).

[15] Little, J. D. C., K. G. Murty, D. W. Sweeney, and C. Karel, "An Algorithm for the Traveling Salesman Problem," Operations Research 11, 972-989 (1963).

[16] McKeown, P. G., "An Extreme Point Ranking Algorithm for Solving the Linear Fixed Charge Problem," Ph.D. Dissertation, University of North Carolina (1973).

[17] McKeown, P. G., "A Vertex Ranking Procedure for Solving the Linear Fixed Charge Problem," Operations Research, 23, No. 6, 1183-1191, (1975).

[18] McKeown, P. G., "The Quadratic Transportation Problem," (submitted to Review of Transportation and Logistics).

[19] McKeown, P. G. and D. S. Rubin, "Adjacent Vertices on Transportation Polytopes," Naval Research Logistics Quarterly, 22, No. 2, 365-374 (1975).

[20] Murty, K., "Solving the Fixed Charge Problem by Ranking the Extreme Points," Operations Research, 16, 268-279 (1968).

[21] Murty, K., "An Algorithm for Ranking All Assignments in Order to Increasing Costs," Operations Research, 16, 682-687 (1968).

[22] Murty, K., "On the Tours of the Traveling Salesman," SIAM Journal on Control, 7, 122-131 (1969).

[23] Murty, K. and C. Karel, "The Traveling Salesman Problem: Solution by Method of Ranking Assignments," Case Institute of Technology, 1961.

[24] Oi, W. Y. and A. P. Hurter, Jr., Economics of Private Truck Transportation, Wm. C. Brown Company, Dubuque, Iowa, 1965.

[25] Rubin, D. S., "Neighboring Vertices on Convex Polyhedral Sets," University of North Carolina (1972).

[26] Steinberg, D. I., "The Fixed Charge Problem," Naval Research Logistics Quarterly, 17, 217-235 (1970).

[27] Sweeney, D. W. and T. William, private communication, May, 1976.

[28] Taha, H. A., "Concave Minimization Over a Convex Polyhedron," Naval Research Logistics Quarterly, 20, No. 3, 533-547 (1973).

[29] Tui, H., "Concave Programming Under Linear Constraints," Soviet Mathematics, 5, 1437-1440 (1964).

A NEW ALTERNATING BASIS ALGORITHM FOR SEMI-ASSIGNMENT NETWORKS

by

Richard Barr, Assistant Professor, Southern Methodist University
Fred Glover, Professor, University of Colorado
Darwin Klingman, Professor, University of Texas
BEB 608
Austin, TX 78712

ABSTRACT

During the early 1970's, highly efficient special purpose computer codes were developed for solving capacitated transshipment problems based on primal extreme point algorithms. Computational comparisons of these codes with the best non-extreme point codes designed for the same class of problems indicated that the primal extreme point-based codes were substantially superior both in terms of computer time and memory requirements on all types of network problems. More recently, specialized non-extreme point codes have been developed for uncapacitated bipartite problems--notably assignment and semi-assignment problems. Comparisons of these problem specific non-extreme point codes with the earlier general purpose extreme point codes casts doubt on the earlier computational conclusions. Consequently, the purposes of this paper are to develop a new extreme point algorithm which is specifically designed to take advantage of bipartite, boolean flows, and degeneracy aspects of assignment and semi-assignment problems and, further, to conduct an unbiased comparison of the alternative algorithmic approaches for solving assignment and semi-assignment problems.

1. INTRODUCTION

The semi-assignment problem is a bipartite network problem whose supply constraints are the same as those of an assignment problem and whose demand constraints are the same as those of a transportation problem, or vice versa. In the midst of the dramatic advances [1,2,5,11,14] in network solution technology since 1969, this important member of the network family called the semi-assignment problem has received scant attention. Falling midway between the classical assignment problem and the classical transportation in its generality, it was bypassed alike by those who studied the ultra-specialized assignment structures and those who studied the more general bipartite transportation structures.

The neglect of the semi-assignment problem is especially ironic in view of the fact that it occupies one of the singularly important niches in the network hierarchy. The "assignment half" captures the ubiquitous multiple choice structures of capital budgeting and planning problems and the special ordered set constraints of mixed integer and combinatorial programming. The "transportation half" captures arbitrary upper and lower bounds on disjoint sums of variables, and therefore can provide valid relaxations for any mixed integer program with imbedded multiple choice and special ordered set structures. Still more directly, the semi-assignment structure appears in large scale scheduling and planning problems from real world settings. For example, applications of manpower planning (assigning personnel to jobs), scheduling (assigning aircraft to routes, trucks to routes, freight to transports, etc.), project planning (assigning project components or subassemblies to tasks over time), and a variety of other practical problems in planning logistics contain embedded semi-assignment problems.

Consequently, the purposes of this paper are to develop a new extreme point algorithm (called the alternating basis (AB) algorithm) which is specifically designed to take advantage of bipartite, boolean flow, and degeneracy aspects of assignment and semi-assignment problems and, further, to conduct a comparison of the alternative algorithmic approaches using codes designed for solving these problems.

Computational testing has shown that approximately 90 percent of the pivots within special purpose primal simplex-based algorithms [2,11,13] are degenerate for assignment and semi-assignment problems with more than 1000 nodes. The primal extreme point algorithm presented in the paper for solving semi-assignment problems which both circumvents and exploits degeneracy can be viewed as an extension of the algorithm presented in [4] and a specialization of the algorithm presented in [8]. One of the principal features of this algorithm is a strong form of convergence that limits the number of degenerate steps in a far more powerful way than achieved by "lexicographic improvement," as for example, in customary LP perturbation schemes.

Each basis examined by this algorithm is restricted to have a certain topology. We show that if a semi-assignment problem has an optimal solution, then an optimal solution can be found by considering only bases of this type. The major mathematical differences between the AB algorithm and the simplex method are (1) the rules of the algorithm automatically (without search) assure that all bases have the special topological structure, and bypasses all other bases normally given consideration by the simplex method; (2) the algorithm is finitely convergent without reliance upon "external" techniques (such as lexicography or perturbation); and (3) in certain cases non-degenerate basis exchanges may be recognized prior to finding the representation of an incoming arc. For these reasons, this algorithm has several computational advantages over the highly efficient special purpose simplex-based codes recently developed for solving network problems.

The AB algorithm also has unique computer implementation properties. Specifically, the data required to represent its bases are substantially less than that required for general simplex bases; thus, the computer memory required to store the basis data is less than that of special purpose simplex-based algorithms. The computational results in section 6 dramatically demonstrate the power and efficiency of the AB algorithm over other algorithms for solving assignment and semi-assignment problems.

2. BACKGROUND MATERIAL

An m x n semi-assignment problem may be defined as:

$$\text{Minimize} \quad \sum_{(i,j)\varepsilon A} c_{ij}\, x_{ij}$$

subject to:

$$\sum_{j\varepsilon\{j:(i,j)\varepsilon A\}} x_{ij} = b_i, \quad i\varepsilon I = \{1,2,\ldots,m\}$$

$$\sum_{i\varepsilon\{i:(i,j)\varepsilon A\}} x_{ij} = 1, \quad j\varepsilon J = \{1,2,\ldots,n\}$$

$$x_{ij} \geq 0, \quad (i,j)\varepsilon A$$

where I is called the set of origin nodes, J is called the set of destination nodes, A is the set of admissible arcs, and c_{ij} is the cost of shipping a unit from origin node i to destination node j.

The dual of the semi-assignment problem may be stated as:

$$\text{Maximize} \quad \sum_{i\varepsilon I} R_i b_i + \sum_{j\varepsilon J} K_j$$

subject to:

$$R_i + K_j \leq c_{ij}, \quad (i,j)\varepsilon A$$

where R_i and K_j are called the node potentials of the origin and destination nodes, respectively.

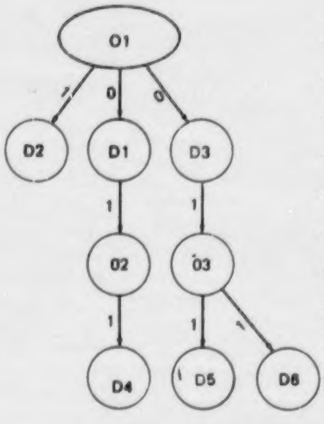
An understanding of the results of this paper relies on a familiarity with graphical interpretations of the semi-assignment problem and how the primal simplex method may be applied to this problem. While these ideas are relatively direct, they, unfortunately, are not succinctly itemized in any references and will be summarized in this section for completeness.

The semi-assignment problem may be represented as a bipartite graph consisting of a set of origin nodes with supplies b_i and a set of destination nodes with unit demands. Directed arcs from origin nodes to destination nodes accommodate the transmission of flow and incur a cost if flow exists. The objective is to determine a set of arc flows which satisfies the supply and demand requirements at minimum total cost.

The bases of the simplex method for solving an m x n semi-assignment problem correspond to spanning trees with m + n - 1 arcs. Exactly n of the basic arcs have an associated basic flow value of one and the other m - 1 arcs have a basic flow value of zero. Therefore each basic solution is highly degenerate (i.e., contains a large number of zero flows). This often causes the simplex method to examine several alternative bases for the same extreme point before moving to an adjacent extreme point.

In the graphical representation approach, the bases of the simplex method for semi-assignment problems are normally kept as rooted trees [5,7,12,14,19]. Conceptually, the root node may be thought of as the highest node in the tree with all of the other nodes hanging below it on directed paths leading downward from the root. Those nodes in the unique path from any given node i to the root are called the *ancestors* of node i, and the immediate ancestor of node i is called its *predecessor*.

Figure 1 illustrates a rooted basis tree, the predecessors of the nodes, and the basic flow values, for a 3 x 6 semi-assignment problem. Notationally, Oi denotes the ith origin node and Dj denotes the jth destination node. The number beside each link (arc) in the basis tree indicates the flow on this arc imparted by the basic solution. Predecessors of nodes are identified in the PREDECESSOR array. For example, as seen from this array, the predecessor of origin node 2 is destination node 1. The root of the tree is node O1 and has no predecessor.



NODE	PREDECESSOR
O1	None
O2	D1
O3	D3
D1	O1
D2	O1
D3	O1
D4	O2
D5	O3
D6	O3

Figure 1--Rooted Basis Tree

It is important to note the direction of the links in Figure 1 correspond to the orientation induced by the predecessor ordering and do *not* necessarily correspond to the direction of the basis arcs in the semi-assignment problem. However, the direction of the basic arcs are known from the bipartite property of the semi-assignment problem; i.e., all problem arcs lead from origin nodes to destination nodes.

In subsequent sections the term *O-D link* and *D-O link* will be used to refer to links in a rooted basis tree that are directed from an origin node to a destination node and vice versa, according to the orientation imparted to the basic arcs by the predecessor indexing. For example, in Figure 1, O2-D4 is an O-D link while D1-O2 is a D-O link. Additionally, basic arcs with a flow of one or zero will be referred to as *1-links* and *0-links,* respectively.

The fundamental pivot step of the simplex method will now be briefly reviewed in the graphical setting. Assume that a feasible starting basis has been determined and is represented as a rooted tree. To evaluate the nonbasic arcs to determine whether any of them "price out" profitably, and therefore are candidates to enter the basis, it is necessary to determine values for the dual variables R_i, $i \in I$, and K_j, $j \in J$, which satisfy complementary slackness; i.e., which yield $R_i + K_j = c_{ij}$ for each basic arc.

There is a unique dual variable associated with each node in the basis tree. For this reason the dual variables--or their values--are often referred to as node potentials. Because of redundancy in the defining equations of the semi-assignment problem (and in network problems generally), one node potential may be specified arbitrarily. The root node is customarily selected for this purpose and assigned a potential of zero, whereupon the potentials of the other nodes are immediately determined in a cascading fashion by moving down the tree and identifying the value for each node from its predecessor using the equation $R_i + K_j = c_{ij}$. Highly efficient labeling procedures for traversing the tree to initialize and update these node potential values are described in [5,12,14].

A feasible basic solution is optimal when all nonbasic arcs satisfy the dual feasibility condition $R_i + K_j \leq c_{ij}$. If the solution is not optimal, then an arc whose dual constraint is violated (i.e., for which $R_i + K_j > c_{ij}$) is selected to enter the basis. The arc to leave the basis is determined by: (1) finding the unique path in the basis tree, called the *basis equivalent path,* which connects the two nodes of the entering arc, and (2) isolating a blocking arc in this path whose flow goes to zero ahead of (or at least as soon as) any others as a result of increasing the flow on the entering arc. In the basis equivalent path, all arcs an even number of links away from the entering arc are called *even* arcs, and all arcs an odd number of links away are called *odd* arcs. An increase in the flow of the incoming arc causes a corresponding increase in the flow of all even arcs and a corresponding decrease in the flow of all odd arcs. Thus, if an odd arc already has a 0 flow, then such an arc qualifies as a blocking arc and the incoming arc cannot be assigned a positive flow.

To illustrate, assume that the starting basis is the one given in Figure 1 and the entering arc is (O3,D4). The basis equivalent path for (O3,D4) is D4-O2-D1-O1-D3-O3. (Note that this path can be easily determined by tracing the chain of predecessors of O3 and D4 to their point of intersection [5,10,12].) As flow is increased on the entering arc, the flow on the odd arc (O1,D1) must be decreased. Since its flow is already zero, (O1,D1) qualifies as a blocking arc so that when arc (O3,D4) is brought into the basis, arc (O1,D1) must be dropped. (There are no other blocking arcs in this case.) In addition, the pivot (or basis exchange) is degenerate since no flow change occurs.

Once the entering and leaving arcs are known, the basis exchange is completed simply by updating the flow values on the basis equivalent path and determining new node potentials for the new basis tree.

Only a subset of the node potentials change during a pivot and these can be updated rather than determined from scratch. This fact will play a crucial role in proving convergence of the algorithm to be developed.

To update the node potentials, assume that the nonbasic arc (p,q) is to enter into the basis and the basic arc (r,s) is to leave the basis. If arc (r,s) is deleted from the basis (before adding arc (p,q)), two subtrees are formed, each containing one of the two nodes of the incoming arc (p,q). Let K denote the subtree which does not contain the root node of the full basis. The node potentials for the new basis may be obtained [12] by updating only those potentials of the nodes in K, as follows. If p is in K, subtract $\delta = R_p + K_q - c_{pq} > 0$ from the potential of each origin node in K and add δ to the potential of each destination node in K. Otherwise, q is in K and $-\delta$ is used in the above operations.

3. ALTERNATING PATH BASIS DEFINITION AND PROPERTIES

The new alternating basis (AB) algorithm for semi-assignment problems developed in this section is similar to the primal simplex method as described above. Its major mathematical distinction is that it does not consider all feasible bases to be candidates for progressing to an optimal basis. That is, the simplex method allows a feasible spanning tree of any structure whatsoever to be included in the set of those that are eligible for consideration as "improving bases" along the path to optimality. However, it will be shown that if a semi-assignment problem has an optimal solution then it also has an optimal solution with a unique basis tree structure, dubbed the alternating path (AP) structure. Furthermore, it will be shown that it is possible to restrict attention *at each step* to bases with this structure. In particular, the AB algorithm is a procedure designed to exploit the properties of the AP basis structure in a manner that substantially reduces the impact of degeneracy, the number of arithmetic operations, and the data storage locations required to solve the semi-assignment problem.

Definition: A rooted basis tree for a semi-assignment problem is an *alternating path (AP)* *basis* if:

1. The root node is an origin node.
2. All 1-links are O-D links.
3. All 0-links are D-O links.

An example of an AP basis is shown in Figure 2.

The "alternating path" designation is applied because every path from a node to any ancestor node in the tree, or vice versa, is an alternating path of 1-links and 0-links. Our attention will chiefly focus on paths from nodes to their ancestors (as would be traced along a succession of predecessors). A path that begins at an origin node and ends at an ancestor destination node will be called "0-AP" because it begins and ends with a 0-link. Similarily, a path that begins at a destination node and ends at an ancestor origin node will be called "1-AP" because it begins and ends with a 1-link.

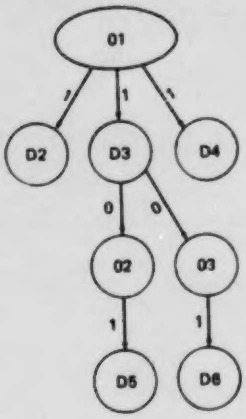


Figure 2--An AP basis for a 3 x 6 semi-assignment problem.

Remark 1: The 1-links of any feasible semi-assignment solution can be augmented by 0-links to create an AP basis (e.g., by adding 0-links from destination nodes to origin nodes in any fashion so that every origin node except the root node has exactly one entering 0-link.) Note if the arcs corresponding to the added 0-links do not exist in the particular semi-assignment problem then a large (big M) cost is assigned such links.

Remark 2: There are many semi-assignment bases for a given feasible solution that are not AP bases. (For example, any basis that has more than one 0-link incident to an origin node is not an AP basis regardless of the origin node chosen as the root. Figure 1 is an example of such a basis.)

Remark 3: An artificially feasible AP basis may always be constructed for an m x n semi-assignment problem by assuming that arcs exist from each origin node to all destination nodes where the non-admissible (artificial) arcs have a "big M" cost. the procedure is as follows.

Initially set $J' = \emptyset$, $B = \emptyset$, and $i = 1$. Go to step 1.

1. Let r be a destination node such that $c_{ir} = \min_{j \in J - J'} c_{ij}$ Set $B = \{(i,r)\} \cup B$, $x_{ir} = 1$, $J' = J' \cup \{r\}$, and $b_i = b_i - 1$.
2. If $b_i \neq 0$ go to step 1. Otherwise, go to step 3.
3. If $i \neq m$, set $i = i + 1$ and go to step 1. Otherwise set $i = 2$, $J' = \emptyset$ and go to step 4.
4. Let r be a destination node such that $c_{ir} = \min_{j \in J - J'} c_{ij}$ and $(i,r) \notin B$ Set $J' = J' \cup \{j:(i,j) \in B\}$, $B = \{(i,r)\} \cup B$, and $x_{ir} = 0$
5. If $i \neq m$, set $i = i + 1$ and go to step 4. Otherwise go to step 6.
6. Using B, create a spanning tree rooted at node 1. The resulting spanning tree will be an AP basis.

Proof:

The remark follows by construction.

Definition: Relative to any AP basis, a nonbasic arc is called a *downward arc* if it connects a destination node to an ancestor origin node, an *upward arc* if it connects an origin node to an ancestor destination node. An arc that connects an origin node and a destination node that do not have either of these ancestral relationships is called a *cross arc*. (Note that these are the only three possibilities for a nonbasic arc in a bipartite network.)

The next two remarks point out some important properties that can be exploited when applying the simplex method to an AP basis.

Remark 4: When the simplex method is applied to an AP basis, a pivot is nondegenerate if and only if the entering nonbasic arc is a downward arc.

Proof:

The remark relies on the fact that a nondegenerate pivot causes the flows on the basis equivalent path to decrease and increase in a strictly alternating fashion to the odd and even links. The "if" part of the remark then follows by observing that a downward arc is 1-AP. The "only if" part of the remark follows from two observations, first that an upward arc is 0-AP, and second that a cross arc has a 0-link above the origin node incident to the entering arc (and this arc is contained in the basis equivalent path adjacent to one of the nodes of the entering arc).

Remark 5: When the simplex method is applied to an AP basis, the pivot can be carried out to give a new AP basis for any entering nonbasic arc simply by dropping the unique link in the basis equivalent path attached to the origin node of the entering arc.

Proof:

The remark follows by observing that an AP basis results if a rooted tree is constructed with its root node equal to the root node of the old AP basis.

Alternating Basis (AB) Algorithm

On the basis of the preceding remarks, the rules of the AB algorithm can be stated in an extremely simple fashion.

1. Select any feasible AP basis for the semi-assignment problem (e.g., using Remark 3).
2. Successively apply the simplex pivot step keeping the root node fixed and picking the link to leave according to Remark 5.

By means of these rules, the foregoing observations imply that the AB algorithm will proceed through a sequence of AP bases, bypassing all other basis structures. Further, these remarks show that a "next" AP basis is always accessible to a given AP basis, so that the method will not be compelled to stop prematurely without being able to carry out a pivot before the optimality (dual feasibility) criteria are satisfied. The issue to be resolved then, is whether the method may progress through a closed circle of AP bases without breaking out, and thus fail to converge. It will be shown that this cannot happen, and that, in fact, the AB algorithm is finitely converging without any reliance upon "external" techniques such as perturbation, as in the ordinary simplex method. Moreover it will be shown that the form of convergence of the AB algorithm has a particularly strong character, in which origin node potentials and destination node potentials each change in a uniform direction throughout any sequence of degenerate pivots.

These results do not require any restrictions on the choice of the incoming variable. For example, it is not necessary to cull through pivot possibilities in an attempt to find degenerate pivot candidates. The following lemma and theorem validate these statements.

Lemma: A basis exchange with the AB algorithm gives rise to a new AP basis in which the new node potentials satisfy the following properties:

a) For a nondegenerate pivot: The changed origin node potentials strictly increase and the changed destination node potentials strictly decrease.

b) For a degenerate pivot: The changed origin node potentials strictly decrease and the changed destination node potentials strictly increase.

Proof:

As already discussed, the node potential values that change may be restricted to those associated with the subtree K. By this procedure, if subtree K contains the origin node of the entering arc then all the origin node potentials in K are decreased and all destination node potentials in K are increased. The reverse is true if the destination node of the entering arc is in subtree K. The lemma then follows from Remarks 4 and 5, which imply that subtree K always contains the destination node of the entering arc for a nondegenerate pivot and the origin node of the entering arc for a degenerate pivot.

Our main result may be stated as follows:

Theorem:

The AB algorithm will obtain an optimal solution (or determine that the problem is infeasible) in a finite number of pivots, regardless of which dual infeasible arc is chosen to be the entering arc, and without any reliance on perturbation or lexicographic orderings.

Proof:

It is sufficient to show that the number of degenerate pivots that occur between any two nondegenerate pivots must be finite. This follows from the second half of the lemma. Note that the node potential assigned to the root node never changes when the node potentials in subtree K are updated. Thus given the constant node potential for the root, the other node potentials are uniquely determined for each successive basis (regardless of the procedure by which they are generated), and the uniform decrease of origin node potentials and the uniform increase of destination node potentials (for the potentials that change) implies that no basis can ever repeat during an uninterrupted sequence of degenerate pivots. This completes the proof.

4. COMPUTATIONAL CONSIDERATIONS

Some of the unique computational features of the AB method include:

a) It explicitly bypasses all "non-AP" basis solutions without requiring any imbedded search procedure or computational tests.

b) It allows degenerate pivots to be recognized and performed without computing the representation of the entering arc. This can be accomplished by using the "cardinality function" of Srinivasan and Thompson [19] which indicates the number of nodes in the subtree of the basis tree below a given node. In particular, following the labeling ideas recently proposed in [5], upward arcs and some cross arcs can be detected simply by comparing the cardinality function values of the nodes associated with the entering arc. That is, denote the cardinality function by f and the entering arc by (p,q). If $f(p) < f(q)$ then arc (p,q) is either an upward arc or a cross arc. In either case the pivot is degenerate and no flow updating is required. Remark 5, furthermore, directly specifies the link to leave the basis. Thus a degenerate pivot simply involves checking the cardinality function, inserting and deleting the appropriate links, and updating the node potential values.

c) Similar streamlining can be achieved for all other pivots. Specifically, if $f(q) < f(p)$ then the appropriate step is to find the first node z on the path from q to the root node such that $f(z) \geq f(p)$. If $z \neq p$ then arc (p,q) is a cross arc and thus the pivot may be executed as before. If $z = p$ then the arc (p,q) is a downward arc and the pivot is nondegenerate. Note that it is only in the case of a nondegenerate pivot that the entire basis equivalent path of the entering arc is traversed. Thus it is only in this case that the complete representation of the entering arc is computed. This is, of course, substantially different than for standard network methods.

Steps b and c above can also be accomplished by using the "distance function" proposed by

Srinivasan and Thompson [19] which indicates the number of links in the basis tree between a given node and the root node. In particular, the distance function may be used in place of the cardinality function by reversing all the above inequalities.

d) Flow values on the basis links never have to be checked to determine the type of pivot. In fact, the structural property of an AP basis, whereby all 1-links are O-D links, makes it unnecessary to store or update any flow values.

e) All of the above computational features may be further enhanced by observing that it is not necessary to store and update a "full" basis tree. That is, while the predecessors are required for each node, the cardinality function values and the "thread" pointers (commonly used to traverse basis subtrees [5,7,14] for node potential updates) need be kept for only the origin nodes. Using this observation, the AB algorithm's basis data storage requirements are roughly half that of the most efficient implementation involving specializations of the simplex method. Moreover, the compression may be used to greatly reduce the number of nodes traversed at each iteration when updating the potentials. By maintaining node potentials for only the origins and saving the unit costs for the arcs represented by the destinations and their predecessors, all data necessary for the pricing operation is available. The destination node potential K_j may be computed as needed using the complementary slackness relationship $K_j = c_{ij} - R_i$. This partial update of the node potentials was originally proposed and tested by Harris [15] for the simplex method as applied to rectangular transportation problems. (This approach has also been tested more recently for transshipment networks by Bradley, Brown, and Graves [7], who have confirmed Harris's findings of its practical merit.) Harris's proposal, however, differs from the above in that the thread pointer and potential update operation is eliminated for those destination nodes of the basis tree which have no descendents. Consequently, Harris's subtrees are twice as large as in our proposal and therefore involve twice the updating effort. In the AB algorithm, maintenance of only the thread pointers and node potentials associated with the origins does not degrade the efficiency of other parts of the algorithm. This is due to the unique structural properties of the AP basis.

5. DEVELOPMENT OF THE AB COMPUTER CODE BY SUBROUTINE

The computer code was written in FORTRAN IV, is an incore code, and was initially tested using the run compiler on a CDC 6600 with a maximum memory of 130,000 words. In this code, a semi-assignment problem with M origins, N destinations, and A arcs (without exploiting the word size of the machine) requires 4M + 2N + 2A + 5000 words. It would be possible by exploiting the fact that the costs, node numbers and node potentials are integer-valued, to store more than one per word and in this manner reduce these storage requirements. However, our purpose was to develop a code whose capabilities did not depend on the unique characteristics of a particular computer (e.g., word size, etc.). The obvious advantage of this approach is the ease with which it enables the code to be tested on different machines. Further, we used a "manilla" FORTRAN IV so that recoding to fit differing machine conventions would be minimized. Within these constraints, we tried to minimize our storage requirements, at the same time making sure the code could solve the "thoroughly general" semi-assignment problem. The code uses the predecessor [10], thread [14], and distance [19] functions to maintain and update the basis data.

A variety of start procedures could be used to find a starting AB basis. However, due to severe time pressure, we only implemented the start procedure of Remark 3.

An important factor influencing computational efficiency is the basis change criterion. The relevant tradeoffs for the basis change criterion involve time consumed in searching for a new arc to enter the basis and the number of pivots required to find an optimal solution (time per pivot versus total number of pivots). Computational testing [2,7,11,13,18,19] has shown that the correct pivot criterion can reduce solution time by as much as a factor of three. Unfortunately, we did not have time to test alternative pivot criteria. The code simply uses the row most negative rule, which was found to be the best in the studies [13,19] for small problems. This criterion scans the arcs of each origin until it encounters the first origin containing a dual infeasibility and then selects the arc of this origin which violates dual feasibility by the largest amount to enter the basis.

The program consists of a main program and three subroutines. The total time spent in each subroutine was recorded by calling a Real Time Clock (accurate to a hundredth of a second) upon entering and leaving that subroutine. A count was also made of the number of nondegenerate and degenerate pivots performed. In the following section, we discuss total solution time (exclusive of input and output), the start time, the number of nondegenerate and degenerate pivots, the total pivot time, and the average pivot time (total pivot time divided by the number of pivots). This code will henceforth be referred to as SA-AB code, for semi-assignment AB algorithm code.

6. COMPUTATIONAL COMPARISON AND CODE REQUIREMENTS

6.1 Computational Comparison of Several Codes

Originally we planned to compare the best version of the SA-AB code (i.e., the code resulting after testing alternative start and pivot rules) with other codes which are based on other algorithms. Unfortunately, we did not have time to test alternative start and pivot procedures. Thus, the following code comparison is a worst case comparison. That is, it is comparing a computationally unimproved version of the SA-AB code with the best version of other codes.

The codes which we obtained for comparison include Bennington [6], BSRL, General Motors, SHARE, and SUPERK [3]. All of these codes are variants of the out-of-kilter method except for Bennington [6]. In addition, the special purpose primal simplex codes ARC-II [5], PNET-I [11], and SUPERT-2 [2] were available for testing. Further, the special purpose dual simplex DNET [11] was available for testing.

The BSRL code was developed by T. Bray and C. Witzgall at the Boeing Scientific Research Laboratories, and the General Motors (GM) code was developed by Rand Corp. and is distributed by the SHARE user group.

All of these codes are in-core codes, i.e., the program and all of the problem data simultaneously reside in fast-access memory. They are all coded in FORTRAN and none of them (including the special purpose primal and dual simplex codes) have been tuned (optimized) for a particular compiler. It is important to note that all the codes except for SUPERT-2 and SA-AB codes are designed to solve capacitated transshipment problems and are not specifically designed to exploit the special structure of semi-assigment problems. Further, SUPERT-2 is designed to solve any uncapacitated transportation problem. All of the problems were solved on the CDC 6600 at the University of Texas Computation Center using the RUN compiler. The computer jobs were executed during periods when the machine load was approximately the same, and all solution times are exclusive of input and output; i.e., the total time spent solving the problem was recorded by calling a Real Time Clock upon starting to solve the problem and again when the solution was obtained.

In addition to these codes, the article by Hatch [15] compares a primal-dual code PD-AAL developed by Decision Systems Associates (DSA) against PNET-I on five assignment problems generated by NETGEN [17]. Since the DSA code is proprietary, we could not obtain a copy of it for comparison. However, with their published times on a CDC-6600, we felt that some comparison could be made if we ran the same problems on a CDC-6600. Thus, in order to compare our results with the results of [15], we solved the same five assignment problems. These results are contained in Table I. When comparing these results, it is important to note that we are not sure that the code PD-AAL is an all FORTRAN code and it is our understanding that the code is fully optimized to exploit the special hardware features of the CDC-6600. This type of specialization could easily increase the performance of all the other codes by a factor of 2 or 3.

A noteworthy feature of the computational results is that SA-AB, PD-AAL, SUPERT-2, and ARC-II are in this order superior to the other codes. Based on the sum of the solution times, SA-AB is roughly fifteen percent faster than its closest competitor and is roughly fifty percent faster than its next closest competitor. Further, the computational results indicate that the AB algorithm reduces the number of pivots by 25% over the simplex algorithm. Additionally, the results indicate that the AB algorithm not only reduces the number of degenerate pivots but also reduces the number of nondegenerate pivots performed on the denser problems. Moreover, the reduction in the nondegenerate pivots (that is, the number of extreme points visited) versus the simplex algorithm increases as the number of arcs increases.

These results lead us to believe that the AB algorithm may be the fastest algorithm for solving assignment problems.

Further, these results indicate that this first implementation of the AB algorithm is 1 1/2 times as fast as the fastest uncapacitated primal simplex transportation code SUPERT-2. Historically, it has always been possible to improve the solution speed of the first implementation of an algorithm by a factor of 2 or 3. Thus, coupling this with the fact that the start and pivot rules of SA-AB have not been computationally investigated, it appears that the AB algorithm is probably twice as fast as other algorithms for solving assignment problems. This result is extremely important since it completely contradicts the older folklore that primal-dual and out-of-kilter algorithms are the fastest and the more recent folklore that special purpose primal simplex based codes are the fastest.

Looking at the out-of-kilter and dual simplex codes' solution times, it is interesting to note that these solution times are much slower than the other codes; further, their times are much more dependent on the number of arcs (holding all other parameters of the problem constant) than the other codes. Another important result which can be gleaned from Table I is that the dual simplex method is not competitive with any of the other algorithms.

After comparing all of the codes on the assignment problems, we choose from available codes the three fastest ones (note that the proprietary PD-AAL code was not available) to compare on semi-assignment problems. In distinction to the assignment test problems, the specification of the semi-assignment test problems vary greatly in both the number of nodes and arcs. As shown in Table II, the eleven test problems vary in size from 50 origins and 500 destinations to 400 origins and 4000 destinations. The number of arcs varies from 2000 arcs to 20,000 arcs. The cost range of the test problems is 1-1000.

The solution times in Table II again indicate that the AB algorithm is substantially superior to the simplex algorithm. The SA-AB times strictly dominate the times of the other codes. Comparing the sum of the total solution times, the SA-AB code is 2.55 times faster than one of the fastest simplex transportation codes, SUPERT-2. This is a rather startling result since some people have indicated that they believe that the speed of the simplex based codes are approaching the computational limits of these problems.

Another noteworthy feature of the computational results is that the SA-AB code is decidely superior on the largest test problems. Consider the last two test problems which each have 400 origins and 4000 destinations. On these problems, the SA-AB is a full three times faster than the SUPERT-2 code.

Based on the results of this testing, the fact that the SA-AB code is the first implementation of the AB algorithm, and the fact that the SA-AB code has not been computationally refined (i.e., alternative start and pivot rules were not examined), we believe that the AB algorithm is currently the fastest algorithm for solving both assignment and semi-assignment problems. This result is extremely encouraging since we have recently extended the concepts of this algorithm

TABLE I

TOTAL SOLUTION TIMES ON 200 X 200 ASSIGNMENT PROBLEMS (IN SECONDS) ON A CDC 6600 WITH A COST RANGE OF 1-100

No. of ARCS	1500	2250	3000	3750	4500	Sum of Times
ARC II	1.45	1.95	2.47	2.67	3.13	11.67
BENN	17.44	20.31	24.92	27.40	--	--
BSRL	30.39	22.08	20.02	23.11	21.08	116.68
DNET	19.87	26.58	27.98	30.15	31.57	136.15
GM	35.67	28.43	31.39	18.62	23.48	137.59
PD-AAL	1.63	1.14	1.89	1.29	1.80	7.75
PNET-I	2.31	3.71	3.47	3.44	4.79	17.72
SA-AB	.97	1.12	1.48	1.61	1.68	6.86
SHARE	19.93	21.17	25.81	24.95	27.05	118.91
SUPERK	6.44	6.47	7.25	6.95	7.56	34.67
SUPERT-2	1.26	1.57	1.98	2.17	2.53	9.51

TABLE II

TOTAL SOLUTION TIMES ON SEMI-ASSIGNMENT PROBLEMS (IN SECONDS) ON A CDC 6600 WITH A COST RANGE OF 1-1000

No. of Nodes m x n	No. of Arcs	ARC-II	SA-AB	SUPERT-2
50 x 500	2,000	2.37	1.14	2.85
50 x 500	5,000	3.53	2.29	3.51
50 x 500	10,000	6.56	4.00	5.53
50 x 1000	4,000	4.27	2.75	6.15
50 x 1000	10,000	9.34	5.64	11.64
50 x 1000	20,000	DNR	8.17	17.02
100 x 1000	4,000	5.59	3.20	6.22
100 x 1000	10,000	10.25	5.62	10.99
100 x 1000	16,000	15.40	8.16	14.27
400 x 4000	10,000	DNR	21.47	61.13
400 x 4000	16,000	DNR	27.81	91.27
SUM OF TOTAL TIMES		-	90.25	230.58

DNR--Did not run as a result of memory limitations.

TABLE III

CODE SPECIFICATIONS

	Developer	Name	Type	Number of Arrays
1.	Barr	SUPERT-2	Primal Simplex Transportation	5 (M + N) + 2A
2.	Barr, Glover, Klingman	ARC-II	Primal Simplex Network	7 (M + N) + 2A
3.	Barr, Glover, Klingman	SA-AB	AB Algorithm	4M + 2N + 2A
4.	Barr, Glover, Klingman	SUPERK	Out-of-kilter	4 (M + N) + 9A
5.	Bennington	BENN	Non-simplex	6 (M + N) + 11A
6.	Bray and Witzgall	BSRL	Out-of-kilter	6 (M + N) + 8A
7.	Clasen	SHARE	Out-of-kilter	6 (M + N) + 7A
8.	Decision System Associates	PD-AAL	Primal-Dual	Not available
9.	Glover, Karney, Klingman	DNET	Dual Simplex network	7 (M + N) + 2A
10.	Glover, Karney, Klingman, Stutz	PNET-I	Primal Simplex network	6 (M + N) + 2A
11.	General Motors	GM	Out-of-kilter	3 (M + N) + 5A

M--Origin Length
N--Destination Length
A--Arc Length

to arbitrary capacitated transportation, transshipment, and generalized transshipment problems.

6.2 Memory Requirements of the Codes

Table III indicates the number of origin, destination, and arc length arrays required in each of the codes testing for solving assignment and semi-assignment problems except for the PD-AAL code. The storage requirements of this code were not available. It should be noted that memory requirements of all of the codes tested were quite close (within 1500 words) excluding the array requirements. Thus, the important factor in comparing the codes is the number of origin, destination, and arc length arrays.

Looking at Table III and keeping in mind that any meaningful problem has to have more arcs than nodes, it is clear that the AB, primal simplex, and dual simplex codes have a distinct advantage (in terms of memory requirements) over all of the other codes. Further, this advantage greatly increases as the number of arcs increase. For example, consider a problem which has 10 times as many arcs as nodes. ARC-II, DNET, PNET-I, SUPERT-2, or SA-AB require only about one-half the memory that the best (in terms of memory requirements) of the other codes. This enables the AB and simplex based codes to solve much larger problems than other codes. Further, it is important to note that the AB based code, SA-AB, requires the least amount of memory. Thus, it appears that the AB algorithm is superior to other algorithms both in terms of solution speed and computer storage requirements.

ACKNOWLEDGEMENTS

This research was partly supported by Project NR047-146, ONR Contract N00014-76-C-0383 with Decision Analysis and Research Institute and by Project NR047-021, ONR Contracts N00014-75-C-0616 and N00014-75-C-0569 with the Center for Cybernetic Studies, The University of Texas. The authors wish to acknowledge the cooperation of the staff of The University of Texas Computation Center, The University of Texas Business School Computation Center, and Southern Methodist University Computation Center.

The authors also wish to acknowledge the indispensable assistance of Dr. John Hultz and Mr. David Karney, systems analyst for Analysis, Research, and Computation, Inc., in the solution testing phase of this study.

REFERENCES

1. Analysis, Research, and Computation, Inc., "Development and Computational Testing on Large Scale Primal Simplex Network Codes," ARC Technical Research Report, P.O. Box 4067, Austin, TX 78765 (1974).
2. R. S. Barr, "Streamlining Primal Simplex Transportation Codes," Research Report to appear, Center for Cybernetic Studies, University of Texas, Austin, Texas.
3. R. S. Barr, F. Glover, and D. Klingman, "An Improved Version of the Out-of-Kilter Method and a Comparative Study of Computer Codes," *Mathematical Programming*, 7, 1, 60-87 (1974).
4. R. S. Barr, F. Glover, and D. Klingman, "The Alternating Basis Algorithm for Assignment Problems," Research Report CCS 263, Center

for Cybernetic Studies, University of Texas at Austin, Austin, TX 78712 (1976).

5. R. S. Barr, F. Glover, and D. Klingman, "Enhancements to Spanning Tree Labeling Procedures for Network Optimization," Research Report CCS 262, Center for Cybernetic Studies, University of Texas at Austin, Austin, TX 78712 (1976).

6. G. E. Bennington, "An Efficient Minimal Cost Flow Algorithm," *Management Science*, 19, 9, 1021-1051 (1973).

7. G. Bradley, G. Brown, G. Graves, "Tailoring Primal Network Codes to Classes of Problems with Common Structure," ORSA/TIMS Conference, Las Vegas (1975).

8. W. H. Cunningham, "A Network Simplex Method," Technical Report No. 207, Department of Mathematical Sciences, John Hopkins University (1974).

9. M. Florian and M. Klein, "An Experimental Evaluation of Some Methods of Solving the Assignment Problem," Technical Report No. 41, Operations Research Group, Columbia University, New York (1969).

10. F. Glover, and D. Klingman, "Locating Stepping-Stone Paths in Distribution Problems Via the Predecessor Index Method," *Transportation Science*, 4, 220-226 (1970).

11. F. Glover, D. Karney, and D. Klingman, "Implementation and Computational Study on Start Procedures and Basis Change Criteria for a Primal Network Code," *Networks*, 4, 3, 191-212 (1974).

12. F. Glover, D. Karney, and D. Klingman, "Augmented Predecessor Index Method for Locating Stepping-Stone Paths and Assigning Dual Prices in Distribution Problems," *Transportation Science*, 6, 1, 171-181 (1972).

13. F. Glover, D. Karney, D. Klingman, and A. Napier, "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," *Management Science*, 20, 5, 793-819 (1974).

14. F. Glover, D. Klingman, and J. Stutz, "Augmented Threaded Index Method for Network Optimization," *INFOR*, 12, 3, 293-298 (1974).

15. R. S. Hatch, "Bench Marks Comparing Transportation Codes based on Primal Simplex and Primal-Dual Algorithms," *Operations Research*, 23, 6, 1167-1171 (1975).

16. B. Harris, "A Code for the Transportation Problem of Linear Programming, *JACM*, 23, 1, 155-157 (1976).

17. D. Klingman, A. Napier, and J. Stutz, "NETGEN-A Program for Generating Large Scale (Un) Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems," *Management Science*, 20, 5, 814-822 (1974).

18. J. Mulvey, "Column Weighting Factors and Other Enhancements to the Augmented Threaded Index Method for Network Optimization," Joint ORSA/TIMS Conference, San Juan, Puerto Rico (1974).

19. V. Srinivasan and G. L. Thompson, "Accelerated Algorithms for Labeling and Relabeling of Trees with Application for Distribution Problems," *JACM*, 19, 4, 712-726 (1972).

RECENT DEVELOPMENTS IN VEHICLE ROUTING

Bruce L. Golden
Operations Research Center
M.I.T.
Cambridge, MA 02139

Abstract

The vehicle routing problem has been regarded as a thorny mathematical programming problem for quite some time. However, only recently has this practical problem received widespread attention in the literature and research efforts in this area continue. Largely this stems from the emergence of algorithmic efficiency as a major concern in Operations Research. In addition, the new respectability of heuristic approaches in response to the large number of intractable NP-complete problems has attracted many researchers. In this paper, we survey some of the recent developments in vehicle routing. We focus on broad issues such as, for example, computational experiments with various algorithms. For vehicle routing problems, the number of feasible solutions is a key factor in algorithmic performance. We demonstrate an approach for calculating this number. Finally, variations dealing with arc routing and extensions concerned with scheduling are introduced.

Introduction

The Vehicle Routing Problem (VRP) seems first to have been mentioned by Garvin et al. [14] in 1957 and by Dantzig and Ramser [11] in 1959. However, only recently has this problem received widespread attention in the literature and research efforts in this area continue. Largely this stems from the emergence of algorithmic implementation as a major concern in Operations Research. In addition, the new respectability of heuristic approaches in response to the large number of intractable NP-complete problems such as the knapsack problem and the Traveling Salesman Problem has attracted many researchers. In this paper, we survey some of the historical developments in the vehicle routing literature. The intention is not to be encyclopedic; the surveys by Bodin [5], Christofides [7], Golden [17], and Turner et al. [38] discuss many details which we will not cover here. Rather, we hope to focus on broad issues and recent developments and provide an overview of vehicle routing. In addition, we count feasible solutions to the VRP and touch upon several new directions in vehicle routing research.

Vehicle Routing Problems, sometimes referred to as truck-dispatching problems, are almost always encountered by complex organizations, and reliable procedures for dealing with them are needed. Recently, higher vehicle costs due to increased oil prices and rising truck drivers' salaries have motivated management to study these issues more carefully.

There may be several hundred demand points in and around a city. The Vehicle Routing Problem is to obtain a set of delivery routes from a central depot to the various demand points each of which has known requirements, which minimizes the total distance covered by the entire fleet. Vehicles have capacities and maximum route time constraints. In addition, the fleet of vehicles may be heterogeneous with respect to these characteristics. It should be noted that there are a number of goals involved which include minimizing the number of vehicles required in the fleet, minimizing travel time or travel distance by vehicles, and generally, providing efficient service to the customers as quickly and cost-effectively as possible. The specific objective function we consider explicitly (minimize travel distance) is both reasonable and tractable. All vehicles depart from the central depot, make a tour of a subset of the demand nodes, and return to the central depot.

Examples of Vehicle Routing Problems include municipal waste collection studied by Beltrami and Bodin [3], fuel oil delivery studied by Garvin et al. [14], newspaper distribution studied by Golden, Magnanti, and Nguyen [19], and routing of school buses studied by Newton and Thomas [26]. It should be clear that all of these vehicle routing problems are more or less the same. Operationally the examples may seem different, but theoretically they can be thought of as equivalent.

Proposed techniques for solving problems of this sort have fallen into two classes--those which solve the problem optimally by branch and bound techniques (Christofides and Eilon [8], Eilon et al. [12], and Pierce [30]), and those which solve the problem heuristically (such as Christofides and Eilon [9], Clarke and Wright [10], Dantzig and Ramser [11], Gaskell [15], Gillett and Miller [16], Golden, Magnanti, and Nguyen [19], Holmes and Parker [20], Russell [34], Tillman and Cochran [37], Tyagi [39], and Yellow [41]). In a loose sense, heuristic algorithms represent sets of rules which produce good solutions to given combinatorial programming problems, but not necessarily the best possible (optimal) solutions. Since algorithms which are guaranteed to achieve optimality are viable only for very small problems, most authors prefer to concentrate on the study of heuristic algorithms.

VRP Formulation

Garvin et al. [14], who treated an application in the oil industry, introduced a mixed integer programming heterogeneous fleet problem formulation of the VRP. This formulation has on the order of n^2 constraints where n is the number of nodes.

Alternatively, when the fleet is homogeneous, the VRP may be expressed as the following large-scale linear-integer program.

$$\text{Minimize} \quad \sum_{j=1}^{m} c_j x_j$$

$$\text{Subject to} \quad \sum_{j=1}^{m} \delta_{ij} x_j = 1, \quad i = 1,\ldots,n$$

$$x_j \epsilon \{0,1\}$$

$$\text{where} \quad \delta_{ij} = \begin{cases} 1 & \text{if } i^{th} \text{ customer is on tour } j \\ 0 & \text{otherwise} \end{cases}$$

$$x_j = \begin{cases} 1 & \text{if tour } j \text{ is chosen} \\ 0 & \text{otherwise} \end{cases}$$

c_j = the length of tour j

n = number of nodes

m = total number of feasible tours.

This set partitioning formulation was first given by Balinski and Quandt [2] in 1964. The objective function minimized total distance traveled. Clearly such a formulation is most valuable for very small Vehicle Routing Problems due to the large number of variables (feasible tours). This formulation may be useful as a conceptual tool, whereas the formulation given by Garvin et al. is more explicit and might be better suited for integer programming techniques. In the worst case

$$m = \sum_{k=1}^{n} \binom{n}{k} k! \gg \sum_{k=1}^{n} \binom{n}{k} = 2^n - 1.$$

For example with n = 25, there may be more than 33 million feasible tours. Balinski and Quandt report success in solving VRP's using a cutting plane algorithm with problems for which $n \le 15$ and $m \le 300$.

The set partitioning problem belongs to the NP-complete problem class along with the VRP. A problem is NP-complete if it can be shown to be equivalent to the Traveling Salesman Problem (TSP) in the following sense. If one can be solved optimally by a polynomial algorithm, then both can be solved by polynomial algorithms. A polynomial algorithm is one in which running time is proportional to a polynomial function of the input. It seems likely, however, that all NP-complete problems are intractable and any optimal algorithm must be of at least exponential time complexity.

We present below a complete linear-integer programming VRP formulation with structure more closely related to the fundamental TSP formulation. This formulation was introduced by Golden, Magnanti, and Nguyen [19].

We point out that in the otherwise excellent survey by Turner et al. [38], an incorrect integer programming formulation is given. Unfortunately, their formulation fails to prevent the formation of subtours. We will refer to the following formulation as the VRP formulation:

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{NV} d_{ij} x_{ij}^{k} \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \sum_{k=1}^{NV} x_{ij}^{k} = 1 \ (j = 2,\ldots,n) \tag{2}$$

$$\sum_{j=1}^{n} \sum_{k=1}^{NV} x_{ij}^{k} = 1 \ (i = 2,\ldots,n) \tag{3}$$

$$\sum_{i=1}^{n} x_{ip}^{k} - \sum_{j=1}^{n} x_{pj}^{k} = 0 \ (k = 1,\ldots,NV; \ p = 1,\ldots,n) \tag{4}$$

$$\sum_{i=1}^{n} Q_i \left\{ \sum_{j=1}^{n} x_{ij}^{k} \right\} \le P_k \ (k = 1,\ldots,NV) \tag{5}$$

$$\sum_{i=1}^{n} t_i^k \sum_{j=1}^{n} x_{ij}^{k} + \sum_{i=1}^{n} \sum_{j=1}^{n} t_{ij}^{k} x_{ij}^{k} \le T_k \quad (k = 1,\ldots,NV) \tag{6}$$

$$\sum_{j=2}^{n} x_{1j}^{k} \le 1 \ (k = 1,\ldots,NV) \tag{7}$$

$$\sum_{i=2}^{n} X_{i1}^{k} \le 1 \ (k = 1,\ldots,NV) \tag{8}$$

$$y_i - y_j + n \sum_{k=1}^{NV} x_{ij}^{k} \le n-1 \tag{9}$$

(for $2 \le i \ne j \le n$ for some real numbers y_i)

$$x_{ij}^{k} = 0 \text{ or } 1 \quad (\text{for all } i,j,k), \tag{10}$$

where n = number of nodes

NV = number of vehicles

P_k = capacity of vehicle k

T_k = maximum time allowed for route of vehicle k

Q_i = demand at node i ($Q_1 = 0$)

t_i^k = time required for vehicle k to deliver or collect at node i ($t_1 = 0$)

t_{ij}^k = travel time for vehicle k from node i to node j ($t_{ii} = \infty$)

d_{ij} = distance from node i to node j

$$x_{ij}^{k} = \begin{cases} 1 \text{ if arc } (i,j) \text{ is traversed by vehicle } k \\ 0 \text{ otherwise.} \end{cases}$$

Node 1 represents the central depot. Equation (1) states that total distance is to be minimized. Equations (2) and (3) ensure that each demand node is served by one vehicle and only one vehicle. Route continuity is represented by equations (4), i.e., if a vehicle enters a demand node, it must exit from that node. Equations (5) are the vehicle capacity constraints; similarly, equations (6) are the total elapsed route time constraints. For instance, a newspaper delivery truck may be restricted from spending more than one hour on a tour in order that the maximum time interval from press to street be made as short as possible. Equations (7) and (8) make certain that vehicle availability is not exceeded. Equations (9) are the subtour-breaking constraints.

Since (1) and (4) imply (3), and (4) and (7) imply (8), the equations (3) and (8) are redundant and can be excluded from the model; in any case, the linear-integer program is awesome in size. Typical problems involve hundreds of demand points.

Historical Survey

Dantzig and Ramser [11] were the first researchers to obtain a method for solving the VRP approximately. In 1964 Clarke and Wright [10] extended the Dantzig and Ramser model to consider routing for a fleet of vehicles of varying capacities. The new procedure borrowed the concept of node aggregation from the earlier method and it seemed to yield better results. Undoubtedly, the Clarke-Wright "savings" method is the most widely used and cited vehicle routing algorithm. It involves first evaluating all potential savings $s_{ij} = d_{1i} + d_{1j} - d_{ij}$ from linking two nodes i and j, and then joining those nodes with the highest feasible savings at each iteration. Initially, each node is served individually from the central depot. This heuristic has been analyzed and modified extensively; Gaskell [15] in 1967 experimented with another savings function $\pi_{ij} = d_{1i} + d_{1j} - 2d_{ij}$ with limited success. In the above savings functions, d_{ij} denotes the distance between nodes i and j.

Tyagi [39] in 1968 presented a method which groups demand points in the following very straightforward fashion. Starting with node 2 (node 1 is the central depot) we find its nearest neighbor, say node k, subject to the vehicle capacity restrictions (we assume here that all vehicle capacities are the same). We next find the nearest neighbor to node k, say node j, subject to the capacity restrictions and continue until adding a nearest neighbor would result in a tour exceeding the maximum vehicle capacity. Rules of thumb are specified to minimize the frequency with which a group will consist of only one delivery point, especially, in the case where the delivery is small or the distance from the central depot to this point is more than half the distance from the farthest point to the central depot. Having grouped the delivery points into m tours, the vehicle dispatching problem reduces to m TSP's, one for each tour. Computational aspects of this algorithm are discussed in Golden, Magnanti, and Nguyen [19].

In 1969, Christofides and Eilon [8] and Pierce [30] presented algorithms for finding optimal solutions to small VRP's. Christofides and Eilon present a branch and bound strategy similar to the well-known Little et al. [24] approach for TSP's. Pierce discusses direct-search, combinatorial programming algorithms for a host of truck-dispatching problems. Delivery deadlines, earliest delivery time constraints, carrier capacity constraints, optional deliveries, and generalized objective functions are developed in detail in his ambitious paper.

Yellow [41], in dealing with Euclidean networks, eliminates the need to initially compute the half-matrix of savings values by applying a simple geometrical search technique based on the polar coordinates of the demand points. In this 1970 paper, he modifies the savings method to produce a sequential Clarke-Wright method where we proceed as in Tyagi's algorithm except that maximum savings rather than minimum distance is our criterion. That is, if we have just linked node i to the subtour, we next find the largest feasible savings between node i and a nearby node j, not yet in the subtour.

The excellent book Distribution Management, published in 1971, studies TSP's and VRP's in great detail [12]. Eilon et al. include an "r-optimal" procedure for the VRP which borrows much from Lin's TSP approach [22]. Their procedure begins with a feasible solution and tests perturbations of r arcs at a time to obtain r-optimality. For example, if r = 2 they examine each pair of arcs to see if it can be replaced by another pair such that feasibility is preserved and total distance is decreased. Lin's approach has been extended by Lin and Kernighan [23] in 1973. It should be noted that many of the book's computational arguments are already out of date.

Krolak et al. [21] in 1972 suggested a man-machine interactive approach which has been successful on some larger problems but seems to require an excessive amount of man-machine time. In 1974, Gillett and Miller [16] proposed a "sweep" algorithm for Euclidean networks which ranks and links demand points by their polar coordinate angle. Newton and Thomas have recently investigated the VRP in connection with school bus routing, dealing with a multi-school system [26]. Their approach for each school has been to obtain a near-minimum single trip solution to the relaxed TSP, and then partition the resulting tour into subtours satisfying the VRP constraints [25].

In Beltrami and Bodin's very comprehensive 1974 paper [3], a variety of problems concerning municipal waste collection are explored. For example, they consider the potential benefit of allowing more than one vehicle to visit a site on the same day, each of which services part of the demand at the site. In addition, some pathological features of the Clarke-Wright heuristics are pointed out.

In 1975, the emphasis in vehicle routing was on the development of codes to solve large-scale

VRP's. Orloff, who had previously formulated the General Routing Problem (GRP), extended his model to handle a fleet of vehicles. The GRP concerns finding a minimum cost cycle which traverses every arc and every node in subsets R and Q of the arcs and nodes respectively. We will discuss node routing and arc routing later in this paper. For now we remark that the GRP formalizes this dichotomy and, at the same time, defines a continuum between these two extremes. Node routing problems tend to be much more difficult to solve than arc routing problems. Recently, Orloff and Caprera [29] have exploited this observation in a heuristic algorithm (of the Lin-Kernighan variety) for solving large GRP's. The strategy is to convert required nodes to required arcs whenever possible. In other words, requiring certain major roads in a transportation network to be traversed in a tour limits the degrees of freedom in an algorithm and reduces running times. Large VRP's may be exploited similarly.

In 1974, Russell [33] presented a modified version of the Gillett and Miller algorithm. More recently, Russell [34] has presented a VRP heuristic algorithm for the special case where the number of tours is pre-specified. Also various sequencing and due-date constraints are considered. The algorithm is an extension of the Lin-Kernighan heuristic [23], and has solved a 159 node problem in under 9 minutes on an IBM 370/168. Robbins et al. [32], also in 1975, have assembled a tour construction-tour improvement code which looks quite promising. Initially, a tour is constructed using the Clarke-Wright approach. Next, this tour is improved by Lin's 2-opt procedure [22]. In five of the seven cases cited, the improvement is less than 1%; the largest improvement is 3.2%. A 150 node VRP was solved in about 37 seconds. This techniques has been applied by Vu and Turner [40] to a rural refuse collection problem. The code presented by Golden, Magnanti, and Nguyen [19] performs from one to two orders of magnitude faster than these recently developed codes although it may not be as accurate.

In their paper, the authors emphasize data structures and list processing and present a new implementation of the Clarke-Wright algorithm which is motivated by (i) optimality considerations, (ii) storage considerations, and (iii) sorting considerations and program running time. The Clarke-Wright algorithm is modified in the following three ways:

(1) by using a route shape parameter γ to define a modified savings $s_{ij} = d_{1i} + d_{1j} - \gamma d_{ij}$ and finding the best route structure obtained as the parameter is varied;

(2) by considering savings only between nodes that are "close" to each other;

(3) by storing savings s_{ij} in a heap structure to reduce comparison operations and ease access.

This modified algorithm has been applied to a Newspaper distribution problem containing nearly 600 drop points. The problem was solved in less that 20 seconds of execution time on an IBM 370/168.

Holmes and Parker [20], in 1976, have suggested another improvement to the Clarke-Wright algorithm. The modification entails a progression from one feasible solution to a next, until we find a local minimum with the following property. If we prohibit any one edge in the current solution from appearing in the next solution (by setting its savings value to zero), and reapply the Clarke-Wright algorithm, we cannot improve upon the total distance. In terms of accuracy, this concept of local minimum seems to yield nice results. Computationally, however, this approach is relatively slow.

Vehicle routing algorithms have recently "come of age" in the sense that they are now capable of solving some large-scale real-world problems. As indicated in this survey, the development has been a slow process. We can categorize the chronological stages of growth as follows:

1) early formulations;
2) early algorithms for hand computations;
3) exact algorithms for small problems;
4) more efficient, computerized algorithms with an emphasis on implementation for moderate size problems;
5) second generation codes, much faster--for large problems;
6) real-world applications.

There are, of course, many important variants of the VRP, some of which we will mention, for which no satisfactory solution techniques yet exist.

We remark in closing this section that many of the papers mentioned are no more than variations and combinations of the "savings" method, the "sweep" method, the "nearest-neighbor" method, and the "r-opt" method.

Counting and the VRP

For hard integer programming problems such as the VRP where enumerative techniques are the primary tool for solving even the smallest problems exactly, the number of feasible solutions is a key factor in algorithmic performance. Generally it is impossible to explicitly enumerate and evaluate all the feasible solutions for all but the smallest problems. In this section, we discuss partitions and counting feasible solutions to the VRP.

In order to get an idea of the number of feasible solutions to the VRP, consider a situation where m vehicles are available to serve n demand points. Each vehicle can make no more than k stops due to capacity constraints. The number of feasible solutions gives an indication of the complexity of these difficult combinatorial programming problems. We demonstrate an approach for calculating this number based on a simple recursion formula.

As background, we briefly introduce some notions about partition theory. The theory of partitions is an area of number theory which deals with the representation of integers as sums of other integers.

Definition 1: A partition of a non-negative integer n is a representation of n as a sum of positive integers, called either summands or parts of the partition. The order of the summands is irrelevant.

The partitions of 5 are 5, 4 + 1, 3 + 2, 3 + 1 + 1, 2 + 2 + 1, 2 + 1 + 1 + 1, and 1 + 1 + 1 + 1 + 1. Thus, there are seven partitions of 5. We remark that 0 has one partition, the empty partition, and that the empty partition has no parts. An excellent source on elementary partition theory is Andrews [1].

Theorem 1: Let $P_k(m,n)$ denote the number of partitions of n into exactly m parts, each of which does not exceed k. Then

$$P_k(m,n) = \begin{cases} P_{k-1}(m,n)+P_k(m-1,n-k) & \text{for } k\geq 1, 1\leq m\leq n \\ 1 & \text{for } k\geq 1, m=n=0 \\ 0 & \text{otherwise.} \end{cases}$$

Proof: The partitions of n into m parts with no part exceeding k can be divided according to whether or not k is a summand. If it is not, then there are $P_{k-1}(m,n)$ partitions. If it is a summand, then the remaining sum is n-k, and we divide into m-1 parts not exceeding k, so $P_k(m-1,n-k)$ describes the appropriate number of such partitions. These events are obviously mutually exclusive and collectively exhaustive, and the recursion is obtained. Boundary conditions are as indicated.

Definition 2: Let $\phi_k(m,n)$ be the number of feasible solutions to the VRP with m vehicles, n demand points, and a delivery capacity of k demand points for each vehicle. We assume vehicles are indistinguishable.

Definition 3: Let $S_k(m,n)$ be the set of all sets $s = \{p_1, p_2, \ldots, p_m\}$ such that $p_1 + p_2 \cdots + p_m = n$ and $0 \leq p_i \leq k$.

Theorem 2: $\phi_k(m,n) = n! \sum_{\ell=1}^{m} P_k(\ell,n)$

Proof: $\phi_k(m,n) = \sum_{s\in S_k(m,n)} \binom{n}{p_1 p_2 \cdots p_m} p_1! p_2! \cdots p_n!$

since we must first distribute the n demand points among the m vehicles (we need not use all the vehicles) and then we must order the points for each vehicle. The value of p_i is the number of demand points assigned to vehicle i. So,

$$\phi_k(m,n) = \sum_{s\in S_k(m,n)} \left(\frac{n!}{p_1! p_2! \cdots p_m!}\right)$$

$$(p_1! p_2! \cdots p_m!)$$

$= n! \cdot \{$the number of partitions of the integer n into at most m parts that have no part exceeding k$\}$

$$= n! \sum_{\ell=1}^{m} P_k(\ell,n).$$

The Traveling Salesman Problem is a special case of the VRP with m = 1 and k = n. $\phi_n(1,n)$ is given by $\phi_n(1,n) = n! P_n(1,n) = n!$ For example, when n = 10, there are 3.6×10^6 tours; when n = 50, 3.04×10^{64} tours; when n = 100, 9.3×10^{157} tours.

We can see how combinatorially explosive TSP's can be. From Theorem 2 we learn that for a problem with n demand nodes the VRP has many more feasible solutions. Suppose we have a situation with 6 demand nodes, 4 vehicles, and a capacity of 4 units. There are a total of 6! = 720 TSP tours. The eleven partitions of 6 are shown below:

6
5 + 1
4 + 2
4 + 1 + 1
3 + 3
3 + 2 + 1
3 + 1 + 1 + 1
2 + 2 + 2
2 + 2 + 1 + 1
2 + 1 + 1 + 1 + 1
1 + 1 + 1 + 1 + 1 + 1.

Since $\phi_4(4,6) = 6! \sum_{\ell=1}^{4} P_4(\ell,6)$ and $\sum_{\ell=1}^{4} P_4(\ell,6) = 7$ the number of feasible solutions to the VRP is 7! = 5040. In general, we can determine $\phi_k(m,n)$ easily after first computing $P_k(\ell,n)$ for $\ell = 1,2,\ldots,m$ via Theorem 1.

The Capacitated Chinese Postman Problem

The problem of finding an optimal route for a single vehicle over a network is a common and, as we have seen, very difficult combinatorial problem. Routing problems can be classified as node routing problems, arc routing problems, or general routing problems. Bodin [5] provides a convenient taxonomy for these problems. The problem of visiting all nodes in a network in the minimal amount of time (node routing) is the classical TSP. The problem of covering all arcs of a network while minimizing the total distance traveled (arc routing) is the Chinese Postman Problem (CPP). The General Routing Problem (GRP) on network G = G(N;A) (N is the set of all nodes, A the set of all arcs) is a generalization which includes the TSP and the CPP as special cases. Here we seek the minimum cost cycle which visits every node in subset $Q \subseteq N$ and covers every arc in subset $R \subseteq A$. This paper is primarily concerned with node routing problems. Orloff introduces the GRP and later extends it to handle more realistic problem situations [27], [28], [29].

Applications of arc routing problems include routing of street sweepers, snow plows, household refuse collection vehicles, postmen, the spraying of roads with salt-grit to prevent ice formation, the inspection of electric power lines, gas, or oil pipelines for fault etc. Since in many vehicle routing situations customers are so numerous that identifying them individually becomes cumbersome, we can consider the CPP to be the continuous counterpart to the discrete TSP. Here an arc replaces a number of customers.

In the CPP we assume that all arcs are undirected. Stricker [36] and Christofides [6] consider an extension which we will call "the Capacitated Chinese Postman Problem" (CCPP), which reflects real-life situations more directly. We are given arc demands $Q = [q_{ij}]$ which must be satisfied by vehicles of capacity W. A number of cycles must be formed which traverse every arc, satisfy demands, and yield minimum distance. This is an arc routing version of the VRP. Stricker and Christofides have suggested heuristic procedures for obtaining reasonable solutions to this problem. However, an optimal solution appears to be very difficult to obtain for all but the most trivial problems. We give an integer programming formulation for the CCPP below which indicates the inherent complexity of this problem.

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{p=1}^{NP} c_{ij} x_{ij}^{p} \tag{11}$$

$$\text{subject to} \quad \sum_{k=1}^{n} x_{ki}^{p} - \sum_{k=1}^{n} x_{ik}^{p} = 0 \text{ for } i=1,\dots,n \quad p=1,\dots NP \tag{12}$$

$$\sum_{p=1}^{NP} (x_{ij}^{p} + x_{ji}^{p}) \geq 1 \quad \text{for all } (i,j)\epsilon A \tag{13}$$

$$\sum_{p=1}^{NP} (\ell_{ij}^{p} + \ell_{ji}^{p}) = 1 \quad \text{for all } (i,j)\epsilon A \tag{14}$$

$$x_{ij}^{p} \geq \ell_{ij}^{p} \quad \text{for all } (i,j)\epsilon A \text{ and } p = 1,\dots, NP \tag{15}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \ell_{ij}^{p} q_{ij} \leq W \quad \text{for } p=1,\dots,NP \tag{16}$$

$$x_{ij}^{p} \geq 0 \text{ and integer} \tag{17}$$

$$\ell_{ij}^{p} \epsilon \{0,1\} \tag{18}$$

where NP = the number of available postmen

x_{ij}^{p} = the number of times arc (i,j) is traversed by postman p

$\ell_{ij}^{p} = \begin{cases} 1 & \text{if postman p services arc (i,j)} \\ 0 & \text{otherwise} \end{cases}$

q_{ij} = the demand on arc (i,j)

W = the vehicle capacity.

The objective function (11) seeks to minimize distance traveled. Equations (12) ensure route continuity. By (13), every arc is covered at least once. Equations (14) state that each arc is serviced exactly one time. Equations (15) guarantee that arc (i,j) can be serviced by postman p only if he covers arc (i,j). Every demand is satisfied, by equations (16). For the small example provided by Christofides [6] with 12 nodes, 22 arcs, and 5 postmen, the above formulation requires 329 constraints and 440 integer variables.

Earlier in this paper, we mentioned that Balinski and Quandt formulated the VRP as a set partitioning problem. Similarly, we can view the CCPP as the following set covering problem:

$$\text{Minimize} \quad \sum_{j=1}^{t} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{t} a_{ij} x_j \geq 1 \text{ for all } i$$

$$x_j \epsilon \{0,1\} \qquad \text{for all } j$$

where c_j = the distance associated with tour j

$x_j = \begin{cases} 1 & \text{if feasible tour j is used} \\ 0 & \text{otherwise} \end{cases}$

$a_{ij} = \begin{cases} 1 & \text{if arc i is covered in feasible tour j} \\ 0 & \text{otherwise} \end{cases}$

t = the number of feasible tours.

As with the VRP, the number of feasible tours can be enormous. The CCPP seems to belong to the same problem class as the TSP and the VRP. Any optimal algorithm for its solution will most likely be exponential.

Christofides [6] presents a heuristic algorithm which appears to be efficient and effective. Computational results are presented for graphs with up to 50 nodes and 125 arcs. Running times are less than 30 seconds and the percent deviation from a lower bound on the solution is less than nine in all cases. Details of the algorithm are clarified and an example is worked through in Christofides [6].

We stress the fact that the CPP and CCPP are the arc routing counterparts to the TSP and VRP. A primary conclusion is that the CCPP is, indeed, a practical problem which deserves much more research attention.

Vehicle Scheduling

We note that any logistics system must be composed of both routing and scheduling components. Although the focus of this paper is on the routing aspect (the time dimension has so far been ignored), we now develop a framework for attacking the two problems simultaneously. The problems have been treated sequentially in Golden [18] in terms of the well-known cutting stock problem.

An alternate, and as yet unexplored, means of connecting the routing and scheduling components is through what Simpson [35] refers to as a schedule map. Whereas a route map considers a spatial, geographical network, the schedule map includes a time dimension as well. Simpson reports on models for public transportation systems, such as airline systems, for which this distinction is valuable. In the figures below this distinction is illustrated.

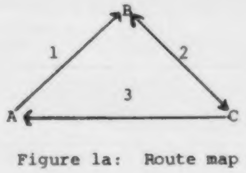


Figure 1a: Route map

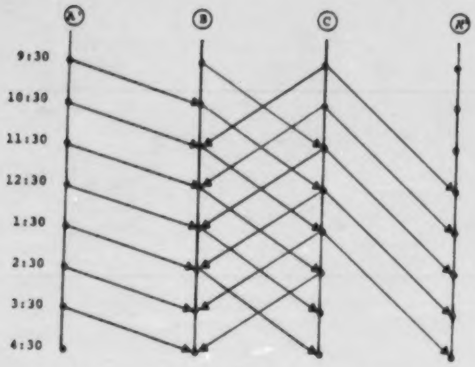


Figure 1b: Schedule map

In the schedule map, each node represents a geographical location and a specific time. We discretize the time axis by dividing the basic unit (perhaps a workday) into a number of smaller units or periods. We can imagine, in the schedule map above, an eight-hour workday broken into hour units. The time points correspond to 9:30, 10:30, 11:30, ..., 4:30. The distances on the route map stand for travel times in terms of these hour units. For example, the travel time from A to B is approximately one hour, from B to C takes approximately two hours, and so on. The number of time points for each location (in this case, eight) is chosen such that the travel times are appropriate. Note that in the schedule map presented, we have split location A into two locations A' and A" in order to make the network more readable.

Once we have constructed a schedule map for the VRP we can proceed to route and schedule simultaneously by applying a sequential Clarke-Wright algorithm. Savings may be defined in terms of travel times between nodes, or in terms of travel distances. When a node i has just been included in a tour, we erase all other nodes which correspond to the same location but different time, and then find another feasible node j (not yet in a tour) which yields the best savings s_{ij}. By feasible, we mean that capacity constraints are not violated. Routes begin from the origin at various time points in conformity with the vehicle availability limitations.

This model leads us to a Generalized Vehicle Routing Problem (GVRP). In the GVRP we are interested in routing vehicles over the collections $S_1, \ldots S_k$ of nodes from a given origin. Each node has a specific demand. We require that at least one node in each collection S_i be serviced and we seek to minimize total distance traveled. The schedule map is one special case of the GVRP. This general problem has applications in many other routing situations where, for instance, there are a number of control offices or warehouses in each small area of a much larger region which must be serviced.

The difficulty with a schedule map is evident. Namely, the number of nodes and arcs to consider grows very rapidly once the time dimension is depicted. In any case, the model itself is helpful for understanding the complex relationship between the routing and scheduling components of a logistics system.

Conclusion

In this paper we have provided a concise overview of vehicle routing. The enormous number of feasible solutions to these integer programming problems becomes a factor in algorithmic performance. We demonstrate how one might determine this number. Two new directions in vehicle routing research have been proposed: the first is an arc routing problem intimately related to the VRP--the Capacited Chinese Postman Problem; the second is a framework for envisioning the interaction between routing and scheduling via the schedule map--The Generalized Vehicle Routing Problem. There are, of course, other important new research areas in vehicle routing such as the evaluation of heuristic algorithms which we have not discussed here.

In brief, the indication is that some of the suggested procedures can be used as effective decision-making tools for large-scale Vehicle Routing Problems encountered in many practical situations in both the public and private sectors.

References

1. G. Andrews, Number Theory, W.B. Saunders Company, Philadelphia (1971).
2. M. Balinski and R. Quandt, "On an Integer Program for a Delivery Problem," Opers. Res., 12 (2), 300-304 (1964).
3. E. Beltrami and L. Bodin, "Networks and Vehicle Routing for Municipal Waste Collection," Networks, 4(1), 65-94 (1974).
4. B. Bennett and D. Gazis, "School Bus Routing by Computer," Trans. Res., 6 (4), 317-324 (1972).
5. L. Bodin, "A Taxonomic Structure for Vehicle Routing and Scheduling Problems," Comput. and Urban Soc., 1, 11-29 (1975).
6. N. Christofides, "The Optimum Traversal of a Graph," OMEGA, 1 (6), 719-732 (1973).
7. N. Christofides, "The Vehicle Routing Problem," presented at the NATO Conference on Combinatorial Optimization, Paris (July 1974).
8. N. Christofides and S. Eilon, "An Algorithm for the Vehicle Dispatching Problem," Opnl. Res. Q., 20, 309 (1969).
9. N. Christofides and S. Eilon, "Algorithms for Large Scale TSP's," Opnl. Res. Q., 23, 511 (1972).
10. G. Clarke and J. Wright, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," Opers. Res., 12 (4), 568-581 (1964).
11. G. Dantzig and J. Ramser, "The Truck Dispatching Problem," Man. Sci., 6, 81-91 (1959).
12. S. Eilon, C. Watson-Gandy, and N. Christofides, Distribution Management, Griffin, London (1971).
13. N. Gartner, B. Golden, and R. Wong, "Modeling and Optimization for Transportation Systems Planning and Operations," Proc. of the Intern. Symp. on Large Engineering Systems, Winnipeg, Canada (Aug. 1976).

14. W. Garvin, H. Crandell, J. John, and R. Spellman, "Applications of Linear Programming in the Oil Industry," *Man. Sci.*, *3* (4), 407-430 (1957).
15. T. Gaskell, "Bases for Vehicle Fleet Scheduling," *Opnl. Res. Q.*, *18*, 281 (1967).
16. B. Gillett and L. Miller, "A Heuristic Algorithm for the Vehicle Dispatch Problem," *Opers. Res.*, *22*, 340 (1974).
17. B. Golden, "Vehicle Routing Problems: Formulations and Heuristic Solution Techniques," M.I.T. Operations Research Center Technical Report No. 113 (August 1975).
18. B. Golden, "Large-Scale Vehicle Routing and Related Combinatorial Problems," Ph.D. Dissertation, Operations Research Center, M.I.T. (1976).
19. B. Golden, T. Magnanti, and H. Nguyen, "Implementing Vehicle Routing Algorithms," M.I.T. Operations Research Center Technical Report No. 115 (September 1975).
20. R. Holmes and R. Parker, "A Vehicle Scheduling Procedure Based Upon Savings and a Solution Perturbation Scheme," *Opnl. Res. Q.*, *27* (1), 83-92 (1976).
21. P. Krolak, W. Felts, and T. Nelson, "A Man-Machine Approach Toward Solving the Generalized Truck Dispatching Problem," *Trans. Sci.*, *6* (2), 149 (1972).
22. S. Lin, "Computer Solutions of the TSP," *Bell System Tech. J.*, *44*, 2245 (1965).
23. S. Lin and B. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Opers. Res.*, *21*, 498-516 (1973).
24. J. Little, K. Murty, D. Sweeney, and C. Karel, "An Algorithm for the Traveling Salesman Problem," *Opers. Res.*, *11* (5), 972-989 (1963) (1963).
25. R. Newton and W. Thomas, "Design of School Bus Routes by Computer," *Socio-Econ. Planning Sciences*, *3*, 75-85 (1969).
26. R. Newton and W. Thomas, "Bus Routing in a Multi-School System," *Computers and Operations Res.*, *1* (2), 213-222 (1974).
27. C. Orloff, "A Fundamental Problem in Vehicle Routing," *Networks*, *4* (1), 35-64 (1974).
28. C. Orloff, "Routing a Fleet of M Vehicles to/from a Central Facility," *Networks*, *4*, 147-162 (1974).
29. C. Orloff and D. Caprera, "Reduction and Solution of Large Scale Vehicle Routing Problems," Princeton University, Transportation Program, Technical Report 75/TR-7 (July 1975).
30. J. Pierce, "Direct Search Algorithms for Truck-Dispatching Problems, Part I," *Trans. Res.*, *3*, 1-42 (1969).
31. J. Robbins, "A Program for Solution of Large Scale Vehicle Routing Problems," Master's Thesis, Dept. of Industrial Engineering and Management, Oklahoma State University (1976).
32. J. Robbins, J. Shamblin, W. Turner, and D. Byrd, "Development of and Computational Experience with a Combination Tour Construction-Tour Improvement Algorithm for Vehicle Routing Problems," presented at ORSA/TIMS Meeting, Las Vegas, Nevada (Nov. 1975).
33. R. Russell, "Efficient Truck Routing for Industrial Refuse Collection," presented at the ORSA/TIMS Meeting, San Juan, Puerto Rico (Oct. 1974).
34. R. Russell, "An Effective Heuristic for the M-Tour Traveling Salesman Problem with Some Side Conditions," presented at the ORSA/TIMS Meeting, Las Vagas, Nevada (Nov. 1975).
35. R. Simpson, "Scheduling and Routing Models for Airline Systems," M.I.T. Flight Transportation Laboratory Report FTL-R68-3 (Dec. 1969).
36. R. Stricker, "Pubic Sector Vehicle Routing: The Chinese Postman Problem," Master's Thesis at M.I.T., Department of Electrical Engineering (1970).
37. F. Tillman and H. Cochran, "A Heuristic Approach for Solving the Delivery Problem," *J. Industrial Eng.*, *19*, 354 (1968).
38. W. Turner, P. Ghare, and L. Fourds, "Transportation Routing Problem--A Survey," *AIIE Transactions*, *6* (4), 288-301 (1974).
39. M. Tyagi, "A Practical Method for the Truck Dispatching Problem," *J. Opers. Res. Soc. Japan*, *10*, 76-92 (1968).
40. V. Vu and W. Turner, "Systems Design for Rural Refuse Collection," *AIIE Transactions* *8* (1), 84-95 (1976).
41. P. Yellow, "A Computational Modification to the Savings Method of Vehicle Scheduling," *Opnl. Res. Q.*, *21*, 281 (1970).

Note

Bruce Golden is now at the University of Maryland, College Park, Maryland.

Balasian-Based Enumeration Procedures: A Study in Computational Efficiency

James H. Patterson
Purdue University

Abstract

Since the introduction of the additive algorithm by Egon Balas in 1965 [1], numerous investigators have added refinements and improvements to the basic procedure to improve the convergence properties of the algorithm. These embellishments have included procedures for changing the search origin, altering the criteria for augmenting variables, etc., and have included the use of composite constraints for fathoming partial solutions early in the augmentation phase of the algorithm. In this paper, we will present computational experience with many of these refinements for the binary as well as the bounded integer problem.

Introduction

A zero-one integer linear programming problem can be written in the form:

$$\begin{aligned} \text{minimize:} \quad & cx \qquad (1) \\ \text{subject to:} \quad & b + Ax \geq 0 \\ & x_j \in \{0,1\} \\ & c_j \geq 0 \end{aligned}$$

where c and x are n-tuples, b is an m-tuple, and A is an m x n matrix. Form (1) is termed the standard form. Any zero-one programming problem can be written in form (1) by a suitable definition of variables and manipulation of constraints.

A bounded integer linear programming problem can be written in the form

$$\begin{aligned} \text{minimize:} \quad & cx \qquad (2) \\ \text{subject to:} \quad & b + Ax \geq 0 \\ & 0 \leq x \leq d \\ & x_j \text{ integer} \\ & c_j \geq 0 \end{aligned}$$

where d is an n-tuple, and the other variables are as defined in (1) above. Suitable transformations of variables and constraints can convert (if necessary) any bounded integer linear programming problem into form (2).

Procedures effective in solving (1) can be used in solving (2) through a binary expansion of the integer variables. Similarly, (1) is a special case of (2) in which $d_j=1$, $j=1,2, \ldots, n$. There exists widespread interest in obtaining solutions to (1) and (2) because of the wide variety of problems which can be formulated as zero-one and bounded integer programming problems (see, for example, [9], [21], [22], [26]).

In this paper, we will evaluate several extensions to the basic Balasian algorithm for solving four classes of integer programming problems. These classes include capital budgeting problems of the Lorie-Savage variety [21], single-constraint allocation problems from Trauth and Woolsey [27], fixed-charge or bounded integer programming problems from Haldi [14], and "IBM test" problems also from Haldi and from Trauth and Woolsey. We will not concern ourselves in this paper with branch and bound procedures of the Land and Doig variety for solving the integer programming problem, as these methods represent fundamentally different approaches. Computational results reported do, however, present the opportunity for comparisons between approaches (depth first search vs. best first search).

Many investigators who have suggested refinements to the Balasian enumeration algorithm have also made available FORTRAN computer programs for testing their refinements [12], [15], [18], [24], [29]. An empirical investigation of each of these improvements remains to be completed.[1] In the next section, the computer programs and modifications examined are briefly described. Each program examined is basically an enumeration algorithm of the Balasian type with certain improvements and modifications to eliminate large portions of the search. Section III discusses the computational experience observed in solving categories of problems with the available techniques. Section IV summarizes the results of the investigation and discusses future improvements to enumeration algorithms which appear promising.

Computer Codes and Modifications Examined

In this section, each of the computer codes (techniques) investigated is briefly described. Each of the codes implements an enumerative search procedure which is a basic Balasian algorithm with modifications designed to exclude portions of the search (implicit enumeration). Each of the codes is redimensioned (upward) from their original versions to fit into 280 K bytes of core. Each code is written entirely in the FORTRAN language and is an "in-core" program. Distinguishing features or characteristics of each of the four codes examined are summarized in Table 1.

Table 1

Computer Codes and Modifications Examined

Code Characteristics	Code				
	RIP30C	DZIP1	DZLP	HOLCOMB	ENUMERS
Language	FORTRAN V (or IV)	FORTRAN V (or IV)	FORTRAN V (or IV)	FORTRAN V (or IV)	FORTRAN V (or IV)
Computer (IBM)	370/168	370/168	370/168	370/168	370/168
In Core	Yes	Yes	Yes	Yes	Yes
Mode of Arrays	Real	Integer	Real (with some double precision)	Integer	Real
Maximum Problem Size, m x n,* (280 K bytes*)	140 x 140	200 x 200	110 x 110	200 x 200	140 x 140
Imbedded L.P. Present	Revised Simplex (with Explicit Inverse)	None	Dual Simplex	None	Revised Simplex
Particular Variation** Examined	1) Imbedded L.P. called every time. 2) One surrogate constraint carried 3) L.P. start used.	1) Branching was determined after preferred variables were selected. 2) Full cancellation test was used. 3) Starting solution always empty.	1) Branching was determined after preferred variables were selected. 2) L.P. applied after cancellation test. 3) L.P. (roundup) determined initial origin 4) One restart used at first solution.	1) More than one variable could enter the basis in an iteration. 2) Starting solution was provided by variable-values which satisfied the sum of all constraints.	1) Bound redefinition employed. 2) L.P called at every iteration. 3) Starting solution always empty. 4) Both Minimum Branch and Balas' Augmentation rules investigated

*These maximum sizes are only approximate; slightly larger sizes may be obtainable. Obviously, there exists a tradeoff in the values of m and n.

**These variations were retained after extensive computational experience was obtained with each code. Most variations are in accordance with authors' recommendations.

RIP30C (Geoffrion and Nelson [12])

The code of Geoffrion and Nelson employs a basic Balasian algorithm and an imbedded L.P. to provide an initial partial solution and to compute a "strongest" surrogate constraint (as defined by Geoffrion [9]). The major advantage of the particular surrogate constraint introduced by Geoffrion and incorporated into RIP30C is that the dual of the required linear program coincides exactly with the continuous version of the zero-one problem in the free variables. Hence, if in computing the composite constraint the partial solution is not fathomed, and if the dual variables are integers (0 or 1), an optimal completion of the partial solution has been found and backtracking can begin in the next iteration. In the computational experience quoted by Geoffrion, it is stated that the "use of the imbedded L.P. greatly reduced solution times in virtually every case." This is opposed to the use of the algorithm without the advantage of the imbedded L. P.

The program is written in general form and makes no advantage of special problem structures. At each iteration a simple test for binary infeasibility and for conditional binary infeasibility is performed, with each constraint being considered individually.

The strategy choices made by the user concern the frequency with which a surrogate constraint is computed and the maximum number of surrogate constraints carried. The L.P. routine used employs the revised simplex method with explicit inverse by Clasen [4].

The only difficulty encountered in using this code concerned the value of ZKBAR input (0). The program looks only for feasible solutions with value at least ZKBAR + .99999 less than the best feasible solution. However, on the IBM 370/168 computer, the code originally did not find the optimal solution on many problems, but terminated at a solution which was always 1 greater than the optimal solution (all c_j's being integer). This difficulty was easily remedied by setting ZKBAR equal to -0.5 on the input card. This difficulty is felt to be machine dependent.

Two versions of RIP30C were investigated: one in which the variables were input in the order given in the problem (RIP30C), and the other in which the variables were initially sorted in increasing order of c_j (SORTED RIP30C). This latter refinement has been found effective elsewhere, and is easily implemented by the user with no modifications to the existing code required.

DZIP1 (Lemke and Spielberg [18])

The code of Lemke and Spielberg employs a direct search technique developed by the authors. On each forward step, one variable is elevated to one; on each backward step one variable is "cancelled" back to 0. Since the algorithm begins with all variables at level 0, the algorithm terminates (in a formal sense) when all variables have been "cancelled" back to level 0.

At each partial solution, the algorithm resolves the 0-1 subproblem in the currently free variables. A backward step is taken whenever it can be established that any permissible forward step leads to an unremovable infeasibility or to a feasible solution not better than the best solution thus far obtained (ceiling criteria). On forward steps, a "preferred set" of the k free variables is constructed by using special types of Gomory cuts and a process termed complete reduction. The preferred variable yielding the maximal Balas value is then elevated to 1. When the subproblem in (k-1) free variables is resolved, the last variable to be elevated to 1 (LIFO procedure) is cancelled to level 0, and the subproblem in the remaining (k-1) free variables is again resolved. The authors state that the procedure of considering a subset of the free variables, the preferred set, reduces substantially the computing time required and the number of points to be considered.

The code uses all integer arithmetic which has a very distinct advantage: half-word integers can be used for the arrays, resulting in a larger problem residing in core or primary memory.

The program performs a change of variables (if necessary) from "problem" to "program" variables in which $0 \le c_j \le c_{j+1}$. This, as well as other elaborations to the basic algorithm, has the effect of significantly reducing computation time. The authors state that the code often finds the optimal, or at best a feasible solution rapidly, with a major portion of the computing time being expended in the "clean-up" phase, where the optimality of the solution is verified.

DZLP (Salkin and Spielberg [24])

The code of Salkin and Spielberg employs a direct search technique which is an elaboration of Balas' additive algorithm. In addition, the code provides for the initial origin to be generated by an imbedded L.P.-roundup start, and also provides for the search to be restarted at improved zero-one solutions. While restarting the search necessarily introduces enumeration redundancy, the authors state that "the length of the search is inversely proportional to the closeness of the origin to the minimal solution" and the redundancy introduced by restarting may be more than compensated for computationally by the reduction in time expended in verifying the optimality of the solution. Computational results reported empirically verifies this contention [23]. The code also contains infeasibility, cancellation, and ceiling criteria similar to that of DZIP1.

The code uses real arithmetic, and, in addition, contains some double precision arrays and variables. The program in the form received required a few minor programming changes to run on the IBM 370/168 computer. Along with a few "carriage control" and other format statement changes, the major change involved correcting the program from returning a "no feasible solution" termination when the first call to the imbedded dual-simplex routine yielded the optimal solution to the zero-one integer problem. Apart from these minor problems, no additional difficulties were encountered in using the program in its original form.

HOLCOMB (Holcomb [15])

The program written by Holcomb at Union Carbide is a variant of the Balasian algorithm with several heuristic tests available for influencing the search strategy. These heuristic tests replace existing tests in the algorithm and allow the user to select a procedure for conducting the search. Heuristic options available include provisions for initializing the program with a starting solution which satisfies a "worst" constraint or which satisfies a surrogate constraint which is the sum of all constraints. Other options available include a provision for entering more than one variable into the basis in an iteration and for raising to 1 the first eligible variable rather than conducting a time-consuming test to determine the next entering variable. Two program strategies are available which are used whenever variations are present in the magnitude of the ratio c_j/a_{ij} or whenever some constraint coefficients a_{ij} are such that $c_j \cdot a_{ij} > 0$.

Best results were obtained using this code to solve the included problems by allowing for more than one variable to enter the basis in an iteration and by providing the program with a starting solution which satisfied the sum of all constraints.

The majority of the arithmetic calculations performed by the code are performed in the fixed-point mode.

ENUMBER8 (Trotter [29])

The ENUMBER8 computer program was developed by Trotter and Shetty [28] to solve the bounded variable integer programming problem using implicit enumeration techniques. Their approach represents an extension to Geoffrion's [9] algorithm for the pure integer problem, employing simple fathoming tests based on optimality and feasibility considerations, and employing surrogate constraints similar to those defined by Geoffrion. The authors also employ techniques to tighten the bounds on free variables using the L.P. based procedure contained in the algorithm, the effects of which are to simultaneously fathom several partial solutions at a time.

As a user option, the augmentation phase of the algorithm (explicit enumeration) may be guided by two mutually exclusive criteria. (1) The maximum BALAS value is used to select that variable x_j to elevate to d_j or its current upper bound. (2) The variable x_j which minimizes d_j for all free variables is fixed at its lowest permissible value.

The authors state that "utilization of the bound redefinition capability appears to offer a general improvement in the performance of the algorithm regardless of the augmentation rule used, although in comparing the computational experience obtained by varying the augmentation rule, the results are less well-defined.

Several difficulties were encountered in using the ENUMBER8 computer code on the IBM computer. These difficulties arose, for the most part, because of 0-subscripting in the arrays, with concomitant problems with several of the indices in DO-loops (being out of range). Several changes thus had to be made in the program, and it is indeed possible that changes made in order to run the program on the IBM computer may have affected the logic originally intended by the authors. Unfortunately, time did not permit us to flow-chart this procedure to examine fully the implications of all changes made to their procedure. We have since used ENUMBER8 on a CDC6500 and a UNIVAC 1110 computer, and have not experienced the difficulty we encountered on the IBM 370.

ENUMBER8 is used in these experiments to assess the efficacy of Trotter and Shetty's approach for solving the pure binary problem (d_j=1, all j), and for comparing their direct approach for treating general integer variables with a binary procedure using a binary expansion to represent the integer variables.

All codes were compiled into object decks using the FORTRAN H compiler with optimization feature (OPT = 2) in order to obtain the lowest execution times possible. The timing subroutine for each code returns relative CPU time in milliseconds using the "T-timer" macro available on the supervisory system.

Computational Experience

Capital Budgeting Problems

The capital budgeting problems given in Table 2 were orginally solved by Petersen [21] in his investigation of variants to the basic Balasian algorithm, some of which were suggested in an earlier paper by Glover. The problems vary in size from six variables and ten constraints to fifty variables and five constraints.

The lowest overall execution time on these seven problems was achieved by SORTED RIP30C, followed by RIP30C and then by DZLP. Both DZIP1 and HOLCOMB began experiencing difficulty in solving these types of problems in the 25-30 variable range and above. Apparently, the addition of the L.P. start and the adaptive origin concept to the ideas originally developed by Lemke and Spielberg possesses great merit in solving these types of problems as is shown in the computation results for DZIP1 and DZLP in Table 2. (DZIP1 and HOLCOMB were unable to solve the seventh and largest capital budgeting problem within the time limit specified.) Sorting of problem variables was also of some importance in reducing execution time for RIP30C. Unfortunately, employing ENUMBER8 with all d_j-values set equal to 1 did not result in low execution times for this group of problems, especially as the number of variables in a problem increased. This corroborates the experience reported by Trotter and Shetty [28] in solving pure binaryproblems with their procedure.

Single-Constraint Allocation Problems

Nine allocation problems were formulated by Trauth and Woolsey to investigate the sensitivity of integer programming algorithms to minor changes in the problem matrix. Each of the problems consists of ten variables. The problems differ from one another only in the value of b, the right hand side. All codes were able to solve these problems in less than 0.10 seconds per problem, hence the results are not presented in tabular form.

IBM Test Problems

The IBM test problems are a pot-pourri of integer programming problems which (except for problem #3) feature matrices of 0's and 1's. Trauth and Woolsey selected these problems for examination because of "the wide disparity of solution times between various approaches and because of the multiplicity of optimum integer solutions." DZIP1 and HOLCOMB experienced difficulty in solving this class of problems when the number of variables increased to 30. The ENUMBER8 programming code recorded its best results when the minimum branch rule was used for variable augmentation (these integer problems were treated directly by ENUMBER8). Overall best results were recorded by the "unsorted version" of RIP30C and by DZLP, as indicated in Table 3.

IBM problems 4 and 5 differ from each other only in the b-vector ($b^5 > b^4$), yet each of the codes examined experienced much more difficulty in solving problem 5 than in solving problem 4. While the difference in solution times cannot be explained by the computational results reported, one can conjecture that constraint severity (as determined by

$$\left(\sum_{i=1}^{m}\left(\sum_{j=1}^{n}\frac{a_{ij}}{b_i}\right)\right)/m$$

in these structured problems has as pronounced an effect on solution times as do the number of variables and/or the number of constraints. The results reported in Table 3 are similar to those reported by Trauth and Woolsey [27, p. 491, Table V], except that each of the codes showed less susceptibility to the large number of constraints in problem 9 than did the integer programming techniques they examined.

Fixed-Charge Integer Programming Problems

The fixed-charge integer programming problems of Haldi were formulated as zero-one integer programming problems by a binary expansion of the bounded integer variables using the binary procedures examined, and were treated directly with the ENUMBER8 program. Each of these problems features special constraints which force certain variables to assume nonzero values if other variables take on nonzero values. Despite the small size of these

Table 2

Solution Times for Petersen's

Capital Budgeting Problems

Problem Number	Number of 0-1 Variables	Number of Constraints	Solution Time**						
			RIP30C	SORTED RIP30C	DZIP1	DZLP	HOLCOMB	ENUMBER8 BALAS	ENUMBER8 MBR
1	6	10	0.050	0.040	0.077	0.040	0.054	0.054	0.060
2***	10	10	0.073	0.077	0.090	0.103	0.067	0.157	0.164
3	15	10	0.174	0.110	0.340	0.130	0.123	0.490	0.507
4	20	10	0.223	0.137	1.727	0.323	0.537	1.457	1.494
5	28	10	0.383	0.257	26.030	1.770	8.407	5.084	5.354
6	39	5	1.706	0.947	46.984	2.337	35.343	10.430	11.464
7	50	5	3.587	2.780	150*	15.917	150*	29.167	31.360
Summary	Number of Problems in Which Optimal Solution Was Found and Verified		7	7	6	7	6	7	7
Measures	Average Solution Time** for Problems Solved		.885	.621	12.541	2.946	7.422	6.691	7.200

*Indicates optimal solution was not found within allotted time of 150 seconds

**IBM 370/168 CPU time, in seconds

***Indicates all c_j's have been multiplied by 10 in this version of the problem in order to have all c_j-values in integer form(to make a valid comparison between computer codes).

problems (in terms of the number of original integer variables and number of constraints), they are quite often difficult to solve using known approaches. Solution times for each of the fixed-charge problems using each of the programming codes investigated are shown in Table 4.

The lowest mean execution time on this series of problems was recorded by DZLP, followed by ENUMBER8 using the minimum branch augmentation rule, then by SORTED RIP30C, RIP30C, HOLCOMB, DZIP1, and ENUMBER8 using BALAS' rule for augmentation. The pure binary programming procedures not employing any form of linear programming to fathom partial solutions generally showed a larger increase in computation times as the number of variables in a problem increased, than did their LP-based counterparts. Also, the ENUMBER8 program with the BALAS augmentation rule was not generally very effective in solving these problems.

It is interesting to compare the results in Table 4 with the integer programming results of Trauth and Woolsey [27, p. 490, Table III] on this series of problems. Problems 1-4 and 7-8 are quite similar, differing primarily in the value of the b-vector and the value of the "fixed-charge." While the codes RIP30C, DZIP1 and, HOLCOMB each record similar computation times for problems within each of these two groups and a marked difference in computing times between the two groups, all were able to solve the problems in a reasonable amount of time. Of the five integer programming techniques examined by Trauth and Woolsey, only one was able to solve all of these problems within the number of iterations allowed. Considering the difference in operating speeds of the computers used, it would appear that problems 1-4 were solved in less time by integer programming. However, all binary programming codes were able to solve each of the fixed-charge problems, and each showed far less variability in solution time between the two groups of problems, 1-4 and 7-8.

In order to compare the direct approach of Trotter and Shetty with a binary expansion of the integer variables and a pure zero-one approach, the fixed charges and the right hand sides for the problems listed in Table 4 were multiplied by 10 and then by 100. For the binary procedures, this resulted in a maximum problem size of 71 variables; for the integer approach, this simply resulted in increasing the bounds on the variables, d_j. Summary results on these larger problems are presented in Table 5. Due to the superiority of the minimum branch rule in the ENUMBER8 program

Table 3

Solution Times for IBM

Test Problems of Haldi

Problem Number	Number of 0-1 Variables	Number of Constraints	Solution Time**						
			RIP30C	SORTED RIP30C	DZIP1	DZLP	HOLCOMB	ENUMBERS BALAS	ENUMBERS MBR
1	21	7	0.294	0.466	0.304	0.186	0.407	0.320	0.226
2	21	7	0.280	0.373	0.594	0.213	0.353	0.647	0.420
3	20	3	0.064	0.106	0.697	0.056	0.090	0.784	0.603
4	30	15	2.947	4.150	36.170	0.213	102.437	2.384	0.903
5	30	15	11.286	12.154	150*	14.283	150*	27.440	10.133
9***	15	50	3.890	4.052	1.604	4.223	1.127	28.830	11.656
Summary	Number of Problems in Which Optimal Solution Was Found and Verified		6	6	5	6	5	6	6
Measures	Average Solution Time** for Problems Solved		3.127	3.550	7.874	3.196	20.883	10.068	3.990

*Indicates optimal solution was not found within time limit of 150 seconds

**IBM 370/168 CPU time, in seconds.

***If one solves the version of this problem as reported by Haldi and by Trauth and Woolsey, the optimal solution is Z=8. If, however, $a_{5,10} = 1$ and $a_{15, 10} = 0$, then the solution reported by Haldi and by Trauth and Woolsey (Z = 9) is correct.

in solving the original version of these problems, this was the only alternative investigated for these larger problems.

Again DZLP recorded the lowest execution times for both problem sizes, and in general, those procedures not using any form of linear programming began to experience significant difficulty in solving these types of problems when the number of variables in the problem increased beyond 30. It is also interesting that the use of a binary expansion for the integer variables when used with a procedure employing linear programming to fathom partial solutions generally resulted in as low if not lower execution times for this group of problems than did a direct treatment of the integer variables. The differences in recorded execution times are, of course, a function of both differences in computer programming decisions aswell as a difference in algorithmic approaches developed.

Summary Observations

The computational experience described in the previous section reveals the efficacy of certain improvements to the basic Balasian algorithm as implemented by computer programs supplied by the authors of these improvements. Any evaluation of these modifications is necessarily confounded with the programming decisions made to implement each improvement, as well as with the programming decisions made regarding the coding of the basic algorithm. Nevertheless, the data does indicate that certain improvements do indeed accelerate convergence on certain categories of problems. The data further provides direction for the user of these techniques for solving certain zero-one and integer programming problems by existing, nonproprietary computer programs,[2] and provides a yardstick for assessing the relative worth of proprietary programs based upon their cost. Additionally, the data suggests certain avenues which might be explored in developing future improvements to known enumeration approaches, some of which have already appeared in the open literature.

Improvements to Existing Techniques

With the exception of perhaps the allocation problems, it appears that any reduction in computation time which could be achieved through the use of integer arithmetic is more than offset by the use of real arrays used in conjunction with an L.P. routine to accelerate convergence. This is evidenced by the lower overall computation times recorded by SORTED RIP30C, RIP30C and DZLP. It would appear, therefore, that any improvement made in the method of solving the imbedded linear program would have the effect of significantly reducing solution times. Geoffrion and Marsten [11] discuss an improved RIP30C[3] which incorporates

Table 4

Solution Times for Fixed Charge Problems of Haldi

Problem Number	Number of 0-1 Variables	Number of Constraints	Solution Time*						
			RIP30C	SORTED RIP30C	DZIP1	DZLP	HOLCOMB	ENUMBER8 BALAS	ENUMBER8 MBR
1	11	4	0.110	0.103	0.090	0.063	0.067	0.070	0.044
2	12	4	0.110	0.137	0.207	0.053	0.053	0.167	0.080
3	14	4	0.150	0.197	0.344	0.100	0.057	0.267	0.114
4	12	4	0.107	0.100	0.166	0.053	0.053	0.314	0.140
7	22	4	0.760	0.440	0.894	0.360	0.510	7.560	0.407
8	23	4	1.173	0.820	1.340	0.340	0.890	8.421	0.527
9	15	6	0.140	0.130	1.454	0.126	0.060	0.907	0.564
10	30	10	1.310	1.783	4.537	0.647	4.970	10.360	0.814
Summary	Number of Problems in Which Optimal Solution Was Found and Verified		8	8	8	8	8	8	8
Measures	Average Solution Time* for Problems Solved		0.483	0.464	1.129	0.218	0.833	3.508	0.336

*IBM 370/168 CPU time, in seconds

Table 5

Mean Solution Times* for Larger

Haldi Fixed Charge Problems

Modification to Original Problem	Mean Solution Time*					
	RIP30C	SORTED RIP30C	DZIP1	DZLP	HOLCOMB	ENUMBER8 MBR
b_i and F_{ij} multiplied by 10	1.320	1.374	6.611	0.840	7.445	1.099
b_i and F_{ij} multiplied by 100	6.328	2.836	35.018	2.775	10.24**	5.366

*IBM 370/168 CPU time, in seconds

**Optimal solution found and verified in only 5 of 8 problems (among other changes) a dual linear programming subroutine with column generation which is purportedly more efficient than the explicit inverse-revised simplex method available in the current version of RIP30C[4]. Given the current state-of-the-art in enumeration programming, it seems safe to conclude that any improved program will necessarily incorporate some type of an imbedded L.P. routine, although the choice of the actual L.P. routine is more open to question. It is worth mentioning in this connection that the problem which led to the discovery that DZLP may on occasion return a "no feasible solution" when the first call to the L.P. returns the optimal zero-one solution was solved in substantially less time by RIP30C with the Clasen L.P. routine than by DZLP.

In general, solution times were lower for the programs which use a starting solution determined by solving the continuous version of the zero-one problem by linear programming and then rounding this solution. Subsequent testing with both RIP-30C and DZLP on the Petersen capital budgeting problems using a null starting vector and a rounded L.P. solution revealed the general efficacy of initializing the implicit enumeration algorithm with other than a vector of zeroes. This would tend to substantiate Salkin's [23] contention that a good initial solution can significantly reduce the length of the search and would further suggest that other initializing schemes be examined. Byrne and Prall [3] have reported success in this area, and the use of a heuristic starting procedure such as the "effective gradient" procedure of Senju and Toyoda [25] also investigated by Wyman [31] shows promise both for initializing the implicit enumeration algorithm and for providing a good initial bound on the objective function value for capital budgeting type problems.

Although less convincing than the general efficacy of employing LP-based routines, the data for the problems examined tend to indicate that the use of a binary expansion for the integer variables may be as effective as treating the integer variables directly, as least within the context of implicit enumeration. Naturally, the differences in results reported is confounded with differences in programming decisions made to implement each of the algorithms described.

Special Problem Structures

Integer and binary programming problems with "special structures" are currently being solved in a modest amount of computation time for problems involving a few hundred variables. Thangavelu and Shetty [26], for example, have developed a special purpose enumeration procedure for solving a binary formulation of the assembly line balancing problem, and quote impressive results. Their procedure has been generalized to the project and job-shop sequencing problem by Patterson and Roth [20], and again good results have been reported. Thus, it is possible that special purpose procedures will, in the future, replace general purpose algorithms on special types of problems as more knoweldge is gained in solving these special structure problems.

Predicting the Time Required for Problem Solution

One of the more important formulations (by application)of binary programming is the capital budgeting problem. Although these types of problems do possess a fairly predictable structure (dense matrices, positive a_{ij}'s, etc.) it is not a structure which is easily exploited. Interest thus centers on whether solution times can be predicted as a function of variables other than n, the number of variables in a problem. The experience gained in solving the IBM problems of Section III suggests that constraint severity as measured by the "amount of slack present in a constraint" can influence the time required for problem solution.

Nine sets of ten capital budgeting problems were computer generated. Each set contained 50,100 or 175 variables and 20 constraints (this would correspond to a planning horizon of 5 years of 4 quarters each, or 20 years of one year each, etc.) Coefficients for the problems were generated randomly such that problems within a set had similar A matrices, but between sets differed in the variability in c_j and b_i values. The problem generator is more fully described in Ference [6]. An attempt was then made to solve the generated problems with the sorted version of RIP30C. These results are summarized in Table 6. In general, those problems possessing the higher variability in b_i values (for similar A-matrices) were solved in significantly less computation time than those with less variability. Subsequent experiments with capital budgeting problems with 11-75 variables indicated the most significant variable in predicting solution time over the range of problem sizes examined was a variance measure of constraint severity given by

$$\left(\sum_{i=1}^{m}\left(\left(\sum_{j=1}^{n}\frac{a_{ij}}{b_i}\right)-\overline{X}\ \mathrm{SEV}\right)^2\right)\div m$$

where $\overline{X}$ SEV =

$$\left(\sum_{i=1}^{m}\left(\sum_{j=1}^{n}\frac{a_{ij}}{b_i}\right)\right)\div m$$

The R-squared values in the regression models developed were on the order of 0.88, indicating it is possible to identify classes of and characteristics of problems which are likely amenable to solution with implicit enumeration techniques.

Conclusion

Pure binary and integer programming problems were input to five different computer codes, each of which incorporates various modifications and improvements to the basic Balasian algorithm. The overall conclusion is that the surrogate constraint concept developed by Geoffrion and programmed into RIP30C and the dynamic origin concept of Salkin and Spielberg are the most effective of the improvements examined,[5] although other improvements were often effective on some problems. The effects of determining good initializing solutions were discussed, and suggestions for incorporating other improvements into zero-one integer programming routines were made. The results reported tend to show that certain

Table 6

Mean Solution Times* Required to Solve Randomly Generated Capital Budgeting Problems

Number of Variables	50				100				175
Variability In c_j - values	Low		High		Low		High		High
Variability In b_i - values	Low	High	Low	High	Low	High	Low	High	High
Mean Solution Time* Ten Problems	37.36	2.146	36.49	1.689	150**	22.78	150**	9.947	38.89

*IBM/370 /168 CPU time, in seconds

**Indicates no problems were solved within time limit of 150 seconds per problem.

types of pure-integer programming problems may be amenable to solution by zero-one integer programming.

Acknowledgments

The author would like to acknowledge the computer programming assistance provided by Joseph J. Albracht, Dean A. Ference, and Walter D. Huber in the preparation of this paper. The anonymous referee provided several suggestions useful in revising the manuscript, as well as the following detailed comments:

1. The special types of cuts employed in the Lemke-Spielberg code (DZIP1), although termed Gomory cuts by the authors, are not the same as those traditionally referenced in the literature as Gomory Cuts, and used in solving the general IP problem (cutting planes).

2. The original use of the generalized origin technique is due to Spielberg, "Plant Location with Generalized Search Origin," Management Science, Vol. 16, No. 3, (November 1969), pp. 165-178

One final comment made by the anonymous referee was that the paper "probably is not up-to-date in the sense that all promising new approaches are included." The intent of the investigation was to evaluate various "depth-first-search" approaches for solving the binary and the bounded integer programming problems using computer software (FORTRAN programs) supplied by the originator of the algorithm evaluated. Within the restrictions of the techniques evaluated and the general availability of software for implementing these techniques, the paper "probably" is fairly up-to-date, although certain other search strategies may offer more promise in obtaining solutions rapidly. Hopefully, the computational experience reported herein can be used as a yardstick for assessing the "promising new approaches" for solving binary and bounded integer programming problems.

Footnotes

1. Investigations of the efficacy of various mathematical programming algorithms for other than the binary programming problem have recently appeared in the open literature. A computational investigation of the "pure-integer" programming problem, for example, is reported by Trauth and Woolsey [27]. Zionts has recently performed some empirical tests of the criss-cross method of linear programming [32]. And Braitsch compares four quadratic programming algorithms in a 1972 paper [2].

2. Each of the programs examined herein is available through SHARE or RAND Corporation at a very modest (usually mailing) cost.

3. The improved RIP30C was not available for testing.

4. Geoffrion (in a private communication) reported solving capital budgeting problems involving more than two-hundred variables with the improved RIP30C.

5. Each of the computer codes examined requires that certain input parameters be set to influence the direction of the search and the search strategy employed. Hence any conclusions made regarding the efficacy of the various approaches should be made in light of the version actually examined. Extreme care was exercised in determining the strategy to be followed; author's recommendations were usually adopted. However, it is possible that a given approach could record lower solution times on a given class of problems using a variant of the solution strategy examined.

Bibliography

1. Balas, E., "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," Operations Research, Vol. 13, No. 4 (July-August, 1965), pp. 517-546.

2. Braitsch, J. R., Jr., "A Computer Comparison of Four Quadratic Programming Algorithms," Management Science, Vol. 18, No. 11 (July 1972), pp. 632-643.

3. Byrne, J. L. and J. G. Prall, "Initializing Geoffrion's Implicit Enumeration Algorithm for the Zero-One Linear Programming Problem," The Computer Journal, Vol 12, No. 4 (November 1969), pp. 381-384.

4. Clasen, R. J., Using Linear Programming as a Simplex Subroutine, The RAND Corporation, P-3267, November 1965.

5. Fleischmann, B., "Computational Experience with the Algorithm of Balas," Operations Research, Vol. 15, No. 1 (January-February 1967), pp. 153-155.

6. Ference, Dean A, "An Analysis of Significant Factors Used In Predicting Solution Times for Capital Budgeting Problems," Unpublished M.B.A. Professional Paper, The Pennsylvania State University, August, 1976.

7. Garfinkel, R. S., and G. L. Nemhauser, Integer Programming, New York: John Wiley & Sons, 1972.

8. Geoffrion, A. M., "Integer Programming by Implicit Enumeration and Balas' Method," SIAM Review, Vol. 9, No. 2 (April 1967), pp. 178-190.

9. _____ "An Improved Implicit Enumeration Approach for Integer Programming," Operations Research, Vol. 17, No. 3 (May-June 1969). p. 137-151.

10. _____ "Elements of Large Scale Mathematical Programming," Management Science, Vol. 16, No. 11 (July 1970), pp. 652-691.

11. _____ and R. E. Marsten, "Integer Programming: A Framework and State-of-the-Art Survey," Management Science, Vol. 18, No. 9 (May 1972), pp. 465-491.

12. _____ and A. B. Nelson, "User's Instructions for 0-1 Integer Linear Programming Code RIP30C." Memorandum RM-5657-PR (May 1968), the RAND Corporation.

13. Glover, F., "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem," Operations Research, Vol. 13, No. 9 (November 1965), pp. 879-919.

14. Haldi, J., "25 Integer Programming Test Problems," Working Paper No. 43, Graduate School of Business, Stanford University, December 1964.

15. Holcomb, B. D., "Zero-One Integer Programming with Heuristics," An IBM contributed program (360D-15.2.011), 1968.

16. Huber, W. D., "Computational Experience with Zero-One Programs for Multi-Project Scheduling," Unpublished M.B.A. Professional Paper, The Pennsylvania State University, August, 1971.

17. Lemke, C. E. and K. Spielberg, "Direct Search Algorithms for Zero-One and Mixed-Integer Programming," Operations Research, Vol. 15, No. 5 (September-October 1967), pp. 892-914.

18. _____ and _____. "DZIP1, Direct Search Zero-One Integer Programming 1," An IBM contributed program (360-D-15.2.001) (corrected) 1968.

19. Patterson, J. H. and W. D. Huber, "A Horizon-Varying, Zero-One Approach to Project Scheduling," Management Science, Vol. 20, No. 6, (February 1974), pp. 990-998.

20. Patterson, J. H. and G. W. Roth, "Project Scheduling under Multiple Resource Constraints: A Zero-One Programming Approach," AIIE Transactions, Vol. 8, No. 3, (December 1976).

21. Peterson, C. ., "Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R & D Projects," Management Science, Vol. 13, No. 9 (May 1967), pp. 736-745.

22. Pritsker, A. A. B., L. J. Watters, and P. M. Wolfe, "Multi-Project Scheduling with Limited Resources: A Zero-One Programming Approach," Management Science, Vol. 16, No. 1 (September 1969), pp. 93-109.

23. Salkin, H. M., "On the Merit of the Generalized Origin and Restarts in Implicit Enumeration," Operations Research, Vol. 18, No. 3 (May-June 1970), pp. 549-555.

24. _____ and K. Spielberg, "DZLP, Adaptive Binary Programming," IBM contributed program (360L-15.0.001), June 1969.

25. Senju, S. and Y. Toyoda, "An Approach to Linear Programming with 0-1 Variables," Management Science, Vol. 15, No. 4 (December 1968), pp. 196-207.

26. Thangavelu, S. R. and C. M. Shetty, "Assembly Line Balancing by 0-1 Integer Programming, AIIE Transactions, Vol. 3, No. 1 (March 1971), pp. 64-69).

27. Trauth, C. A., Jr. and R. E. Woolsey, "Integer Linear Programming: A Study in Computational Efficiency," Management Science, Vol 15, No. 9 (May 1969), pp. 481-493.

28. Trotter, L. E., Jr. and C. M. Shetty, "An Algorithm for the Bounded Variable Integer Programming Problem," JACM, Vol. 21, No. 3 (July 1974), pp. 505-513.

29. Trotter, L. E., Jr. "User's Instructions for the Integer Programming Code ENUMBER8," Mathematics Research Center, University of Wisconsin, - Madison: Technical Summary Report #1347 (December 1973).

30. Wagner, H. M., "An Integer Linear Programming Model for Machine Scheduling," Naval Research Logistics Quarterly, Vol. 6, No. 2 (June 1959), pp. 131-140.

31. Wyman, F. P., "Binary Programming: A Decision Rule for Selecting Optimal vs. Heuristic Techniques," The Computer Journal, Vol. 16, No. 2 (May 1973), pp. 135-140.

32. Zionts, S., "Some Empirical Tests of the Criss-Cross Method," Management Science, Vol. 19, No. 4 (December 1972), pp. 406-410.

A STUDY OF THE EFFECT OF LP PARAMETERS ON ALOGRITHM PERFORMANCE

Christine H. Layman
Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803

Richard P. O'Neill
Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803

Abstract

This paper presents the results of an experiment to determine the relationship between parameters which can be used to describe linear programs and standard measures that can be used to describe the performance of the simplex algorithm. The test problems are generated by LPGENR, a fortran subroutine that generates linear programs, and solved by MPS/360. Problems vary in size from 50 rows and 100 columns to 150 rows, 300 columns and 50 upper bounds with varying densities and solution characteristics. Correlations and least squares fits are calculated to determine the relation between parameters and performance measures. Several unusual relationships are reported.

1. Introduction

Several experimental studies in linear programming have appeared in the literature (see [1], [2], [5], [6]), but much of the experimental work is unpublished and is transmitted as folklore. For example, it is often stated that the number of iterations is between one and three times the number of rows. This study was initiated to examine the relationship between several parameters including some often quoted ones and some performance measures of the simplex alogrithm. At the same time the performance of LPGENR (see [3]) could be examined.

2. Software

LPGENR (see [3] and [8]) is a fortran subroutine system which generates LP problems with prespecified characteristics and a known optimal solution using a pseudo-random number generator. Options allow the parameters to be passed or generated with a specified range. Further, output may be obtained in several forms including MPS card image format. A brief description of the algorithm for generating the problems is given in appendix one.

The generated problems were solved using IBM's MPS/360 with no default changes on the LSU IBM 360/65 under the OS/MVT operating system. This later proved to be a problem since the XCLOCKSW switch was on. This switch causes an INVERT demand to be controlled by the wall clock. Since the operating system is multiprogrammed, reinversion was performed more frequently than desired, sometimes only three iterations apart and in a manner not completely predictable. About 16 problems were rerun and are presented in the appendix.

The statistical analysis was performed by the Statistical Analysis System (SAS) (see [7]).

3. Experiment

The parameters chosen for study were: the number of rows varying from 50 to 150, the number of columns varying from 100 to 300, the matrix density varying from about .40 to .04, the number of upper bounds varying from 0 to half the number of rows. Further, various problem types were included that defined the characteristics of the optimal solution.

The optimal solution characteristics for this study can be divided into three classes: the type of optimal solution, the size of the reduced costs at optimality, and the number of variables at their upper bound. There were three optimal solution types: degenerate, unique with all basic variables greater than zero, and alternative. For degenerate solutions the number of basic variables at zero was set to be twenty-five percent of the number of rows. For alternative optimal solutions the number of nonbasic columns with reduced costs of zero was set to be ten percent of the number of rows. The magnitude of the reduced costs of nonbasic columns at optimality are specified by multiplying the objective coefficient which would produce a zero reduced cost by the factor, 1-r, where r is a uniform random variable over the interval 0 to either 0.1, 0.5 or 1.0. The number of variables at bound in the optimal solution was set to be half the number of upper bounds.

All other input parameters for LPGENR were held constant. Each problem was a maximization with all equality rows. The nonzero entries in the constraint matrix were uniformly distributed between -2 and 10; the optimal nonzero primal and dual solution values from which the cost row and righthand side are generated were uniformly distributed between 0 and 20; all values were rounded to the nearest integer.

The performance measures recorded for each test problem were the number of phase one iterations, the CPU time of the execution step and the percent by which the generated optimal value deviated from the calculated optimal value. The CPU time includes a modest amount of time for setting up the problem.

It was hypothesized that problems in which nonbasic variables at optimality had large reduced costs would solve faster since the optimal columns might be easier for the algorithm to choose. Also, problems with alternative optimal solutions would be easier to find since the algorithm need only find one from many. It is obvious that the density has an effect on the number of computations, but little has been said concerning its effect on the number of iterations.

4. Results

A total of 239 problems were solved and the results are listed in appendix two. In 29 cases, tolerance checks made after reinversion caused the algorithm to terminate. In all but three cases, these can be recognized by a nonzero value in the last column of the table. In five cases (two of the five are replications) indicated by a * in the table, the solution was in fact optimal but in each case the sum of infeasibilities was zero and several degenerate pivots were taken before the problem was declared infeasible after the tolerance check. Some problems, that were rerun after tolerance checks declared them infeasible, continued passed the point where they were previously terminated after reinversion and subsequently, the optimal solution was found and declared as such indicating that in a sense the problem is self-correcting.

Least squares fits and correlation coefficients were calculated for the 50 and 100 row problems that reached the optimal solution. The problems with 150 constraints were not included in these calculations since only three reached optimality and the others encountered numerical problems. The correlation coefficients are listed in table one. Least squares fits were calculated with phase I iterations, phase II iterations, total iterations and CPU time as dependent variables. The calculations are presented for CPU time but not discussed because of the wall clock reinversion demand. All fits were forced to have a zero intercept. The results are presented in tables two, three, four and five. The values in each row are the least squares coefficient of the independent variable and the standard error of the coefficient. The last three variables are the coefficients for the degenerate, unique and alternative optimal solutions which were created as dummy variables.

When total iterations is the dependent variable, the three variables contributing most to the fit are, in order, the number of columns, the number of rows and the density. The coefficient of the number of rows is 2.09 which is consistent with folklore. The number of columns, the independent variable which contributed most to the fit, had a coefficient of 1.3 and the density had a large negative coefficient.

When phase one iterations is the dependent variable, the three variables contributing most to the fit are, in order, the number of rows, the number of columns, and the density. The number of rows is more important in explaining phase one iterations since the criteria is simply to find a feasible solution. Again the density produces a negative influence.

When phase two iterations is the dependent variable, the three variables contributing most to the fit are, in order, the number of columns, the density and the number of upper bounds. Surprisingly the number of rows has a negative coefficient, but contributes very little to the fit indicating that its effect is virtually negligible. The major part of the explanation is due to the number of columns and the density which again shows up negative.

Although the dummy variables that represent the types of optimal solution are not among the variables contributing most to the fits, it is interesting to note some general trends. When comparing the three solution types it is perhaps easiest to consider the differences between the coefficients. In each fit using iterations the difference between the coefficients for degenerate and either unique or alternative solutions indicates that degenerate solutions require more iterations than either unique or alternative optimal solutions.

5. Discussion of the Results

The most surprising results concern the reduced cost perturbation and the density. The different reduced perturbations produced virtually no effect on any of the performance measures in any of the statistical calculations. Another surprising result is the negligible effect of the number of rows on phase two iterations.

Although the density had a positive influence on CPU time, it produced a negative effect on the iteration measures. That is, lower density problems required more iterations than problems with higher density. A problem with lower density could result in more degenerate pivots. The increased iterations when problems have degenerate optimal solutions could also be the result of a greater amount of degenerate pivots on the way to the optimum. Both of these speculations support the folklore that degenerate problems often require more iterations.

To our knowledge this is the first study that has attempted to consider the joint effect of a number of parameters. A question not answered by this study is how generated problems compare to real world problems. One conclusion that can be made is that the generator can be specified to create problems that confuse the algorithm. For example, the problems that were declared infeasible when they were optimal.

Correlation Coefficients

	Phase I iterations	Phase II iterations	Total iterations	CPU time
number of rows	.49	-.31	.13	.42
number of columns	.34	.85	.66	.39
number of upper bounds	.13	.04	.10	.12
density	-.38	-.20	-.33	.01
reduced cost perturbation	.02	-.01	.00	.00

table one

Dependent variable: phase one iterations

$R^2 = .93$

independent variable	coeff.	std. err.
number of rows	2.31	0.16
number of columns	0.55	0.04
number of upper bounds	-0.14	0.20
density	-190	31.5
reduced cost perturbation	2.13	8.82
degenerate optimum	-24.8	18.8
unique optimum	-50.7	18.4
alternative optimum	-43.1	18.7

table two

Dependent variable: phase two iterations

$R^2 = .91$

independent variable	coeff.	std. err.
number of rows	-0.22	0.12
number of columns	0.74	0.03
number of upper bounds	0.32	0.14
density	-106	22.8
reduced cost perturbation	-2.90	6.37
degenerate optimum	2.74	13.6
unique optimum	-20.9	13.3
alternative optimum	-20.5	13.5

table three

Dependent variable: total iterations

$R^2 = .93$

independent variable	coeff.	std. err.
number of rows	2.09	0.26
number of columns	1.30	0.07
number of upper bounds	0.18	0.31
density	-296	49.8
reduced cost perturbation	-0.76	13.9
degenerate optimum	-22.1	29.8
unique optimum	-71.6	29.2
alternative optimum	-63.6	29.6

table four

Dependent variable: CPU time

$R^2 = .84$

independent variable	coeff.	std. err.
number of rows	2.27	0.18
number of columns	0.60	0.05
number of upper bounds	0.005	0.21
density	106	34.2
reduced cost perturbation	-1.89	9.57
degenerate optimum	-160	20.4
unique optimum	-177	20.0
alternative optimum	-159	20.2

table five

References

1. Dickson, J. C. and F. P. Frederick. "A Decision Rule for Improved Efficiency in Solving Linear Programming Problems with the Simplex Algorithm," Communications of the ACM, Sept., 1960, p. 509.

2. Kuhn, H. W. and R. E. Quandt. "An Experimental Study of the Simplex Method," Proc. of the Symposia in Applied Mathematics, American Mathematical Society, 1963.

3. Michaels, W. M. and R. P. O'Neill. "A Mathematical Program Generator," presented at the ORSA/TIMS Metting, Chicago, Ill., April 30, 1975.

4. "Mathematical Programming System/360 Version 2," IBM Corporation. White Plains, N.Y., GH20-0476-2, 1968.

5. Ross, G. T., D. Klingman and A. Napier. "A Computational Study of the Effects of Problem Dimensions on Solution Times for Transportation Problems," Journal of ACM, vol. 22, July, 1975, p. 413.

6. Wolfe, P. and L. Cutler. "Experiments in Linear Programming," Recent Advances in Mathematical Programming, eds. Graves and Wolfe, McGraw Hill, N.Y., N.Y., 1963, p. 177-201.

7. Service, Joloyne. "A User's Guide to the Statistical Analysis System," Dept. of Statistics, North Carolina State Univ., Raleigh, North Carolina, 1972.

8. Michaels, W. M. and R. P. O'Neill. "User's Guide for LPGENR," Dept. of Computer Science, LSU, Baton Rouge, LA, April, 1975.

Appendix One

LPGENR Algorithm for Generating Linear Programs

Given the following linear program

$$\begin{array}{lll} \max & cx & \\ & Ax = b & A \text{ is } m \text{ by } n \\ & 0 \le x \le d & \end{array}$$

LPGENR generates c, A, b and d as follows:

step 1: Generate $x^* \geq 0$, an optimal solution and $(u^*, v^*) \geq 0$, an optimal set of multipliers. A degenerate optimal solution will have more than n-m variables in x^* at 0; a unique optimal solution will have exactly m variables greater than zero; an optimal solution with alternative optima will have more than m variables greater than zero.

```
Step 2:  for $j = 1$ to $n$
         if $v_j^* > 0$, $d_j = x_j^*$
         if $v_j^* = 0$, $d_j = x_j^* (1 + URV[0, .5])$

step 3:  for $i = 1$ to $m$ and $j = 1$ to $n$
         if $URV[0, 1] \leq$ density, $a_{ij} = URV[-2,10]$
         otherwise, $a_{ij} = 0$

step 4:  for $i = 1$ to $m$
         $b_i = A_i x^*$

step 5:  for $j = 1$ to $n$
         if $x_j^* > 0$, $c_j = u^* A_j + v_j^*$
         if $x_j^* = 0$, $c_j = (u^* A_j + v_j^*)(1 \pm URV[0,t])$
         where $\pm$ is determined by the sign of
         $u^* A_j + v^*$ and t is the reduced cost
         perturbation.
```

Note: URV[e,f] is a uniform pseudo random number between e and f.

NUMBER OF ROWS	NUMBER OF COLUMNS	NUMBER OF UPPER BOUNDS
50	100	0

DENSITY	REDUCED COST PERTURB.	OPT. SOLN. TYPE	ITERATIONS PHASE 1	ITERATIONS PHASE 2	ITERATIONS TOTAL	CPU TIME (SEC)	PERCENT DEVIATION FROM OPTIMUM
0.08	0.1	1	107	54	161	27.9	0.0
0.08	0.5	1	107	54	161	28.4	0.0
0.08	1.0	1	102	38	140	25.9	0.0
0.08	0.1	0	143	75	218	36.8	0.0
0.08	0.5	0	143	68	211	36.8	0.0
0.08	1.0	0	143	78	221	38.3	0.0
0.08	0.1	2	100	24	124	24.9	0.0
0.08	0.5	2	100	24	124	23.6	0.0
0.08	1.0	2	100	25	125	25.3	0.0
0.17	0.1	1	112	40	152	40.8	0.0
0.17	0.5	1	112	40	152	39.7	0.0
0.17	1.0	1	112	40	152	39.1	0.0
0.17	0.1	0	149	55	204	50.2	0.0
0.17	0.5	0	149	55	204	48.6	0.0
0.17	1.0	0	149	63	212	49.3	0.0
0.17	0.1	2	113	57	170	43.5	0.0
0.17	0.5	2	113	57	170	45.1	0.0
0.17	1.0	2	113	57	170	44.1	0.0
0.33	0.1	1	95	50	145	56.6	0.0
0.33	0.5	1	95	50	145	55.5	0.0
0.33	1.0	1	95	50	145	53.1	0.0
0.33	0.1	0	120	36	156	56.0	0.0
0.33	0.5	0	120	36	156	58.5	0.0
0.33	1.0	0	120	35	155	55.5	0.0
0.33	0.1	2	111	28	139	49.3	0.0
0.33	0.5	2	111	28	139	49.1	0.0
0.33	1.0	2	111	28	139	50.6	0.0

NUMBER OF ROWS	NUMBER OF COLUMNS	NUMBER OF UPPER BOUNDS
50	100	25

DENSITY	REDUCED COST PERTURB.	OPT. SOLN. TYPE	ITERATIONS PHASE 1	ITERATIONS PHASE 2	ITERATIONS TOTAL	CPU TIME (SEC)	PERCENT DEVIATION FROM OPTIMUM
0.08	0.1	1	126	27	153	25.5	0.0
0.08	0.5	1	126	36	162	27.0	0.0
0.08	1.0	1	126	32	158	26.1	0.0
0.08	0.1	0	103	34	137	26.1	0.0
0.08	0.5	0	103	20	123	23.4	0.0
0.08	1.0	0	103	24	127	24.9	0.0
0.08	0.1	2	113	33	146	28.6	0.0
0.08	0.5	2	113	32	145	27.6	0.0
0.08	1.0	2	113	32	145	30.4	0.0
0.17	0.1	1	107	43	150	43.0	0.0
0.17	0.5	1	107	47	154	43.6	0.0
0.17	1.0	1	107	44	151	43.5	0.0
0.17	0.1	0	135	72	207	59.5	0.0
0.17	0.5	0	135	72	207	57.4	0.0
0.17	1.0	0	135	80	215	52.4	0.0
0.17	0.1	2	125	23	148	49.3	0.0
0.17	0.5	2	125	32	157	49.9	0.0
0.17	1.0	2	125	30	155	44.6	0.0
0.33	0.1	1	111	38	149	55.2	0.0
0.33	0.5	1	111	33	144	58.7	0.0
0.33	1.0	1	111	32	143	64.2	0.0
0.33	0.1	0	116	51	167	60.6	0.0
0.33	0.5	0	116	48	164	61.1	0.0
0.33	1.0	0	116	53	169	61.6	0.0
0.33	0.1	2	117	56	173	67.6	0.0
0.33	0.5	2	117	68	185	63.2	0.0
0.33	1.0	2	117	67	184	70.4	0.0

NUMBER OF ROWS	NUMBER OF COLUMNS	NUMBER OF UPPER BOUNDS
50	200	0

DENSITY	REDUCED COST PERTURB.	OPT. SOLN. TYPE	ITERATIONS PHASE 1	 PHASE 2	 TOTAL	CPU TIME (SEC)	PERCENT DEVIATION FROM OPTIMUM
0.08	0.1	1	182	120	302	74.2	0.0
0.08	0.5	1	182	120	302	69.9	0.0
0.08	1.0	1	182	120	302	74.0	0.0
0.17	0.1	1	164	77	241	80.4	0.0
0.17	0.5	1	164	72	236	79.2	0.0
0.17	1.0	1	164	77	241	80.6	0.0
0.33	0.1	1	127	105	232	119.1	0.0
0.33	0.5	1	127	115	242	119.2	0.0
0.33	1.0	1	127	105	232	113.8	0.0
0.08	0.1	0	150	144	294	61.7	0.0
0.08	0.5	0	150	145	295	63.3	0.0
0.08	1.0	0	150	117	267	59.0	0.0
0.17	0.1	0	190	102	292	92.7	0.0
0.17	0.5	0	190	100	290	93.6	0.0
0.17	1.0	0	190	102	292	92.6	0.0
0.34	0.1	0	111	111	222	108.7	0.0
0.34	0.5	0	111	105	216	106.1	0.0
0.34	1.0	0	111	98	209	107.0	0.0
0.08	0.1	2	170	88	258	54.8	0.0
0.08	0.5	2	162	105	267	57.3	0.0
0.08	1.0	2	170	88	258	55.8	0.0
0.17	0.1	2	140	106	246	85.7	0.0
0.17	0.5	2	140	110	250	84.6	0.0
0.17	1.0	2	140	109	249	92.2	0.0
0.33	0.1	2	110	99	209	115.6	0.0
0.33	0.5	2	110	99	209	116.3	0.0
0.33	1.0	2	110	99	209	107.2	0.0

NUMBER OF ROWS	NUMBER OF COLUMNS	NUMBER OF UPPER BOUNDS
50	200	25

DENSITY	REDUCED COST PERTURB.	OPT. SOLN. TYPE	ITERATIONS PHASE 1	 PHASE 2	 TOTAL	CPU TIME (SEC)	PERCENT DEVIATION FROM OPTIMUM
0.08	0.1	1	110	104	214	47.4	0.0
0.08	0.5	1	110	125	235	54.7	0.0
0.08	1.0	1	110	145	255	57.6	0.0
0.17	0.1	1	219	92	311	99.8	0.0
0.17	0.5	1	219	104	323	104.6	0.0
0.17	1.0	1	219	107	326	99.6	0.0
0.33	0.1	1	151	140	291	138.7	0.0
0.33	0.5	1	151	144	295	144.9	0.0
0.33	1.0	1	151	118	269	136.2	0.0
0.08	0.1	0	164	172	336	64.2	0.0
0.08	0.5	0	164	202	366	67.1	0.0
0.08	1.0	0	165	150	315	65.3	0.0
0.17	0.1	0	159	107	266	89.4	0.0
0.17	0.5	0	159	118	277	91.4	0.0
0.17	1.0	0	159	129	288	85.1	0.0
0.34	0.1	0	142	146	288	140.8	0.0
0.34	0.5	0	142	132	274	138.0	0.0
0.34	1.0	0	142	144	286	134.0	0.0
0.08	0.1	2	158	122	280	59.6	0.0
0.08	0.5	2	158	119	277	57.4	0.0
0.08	1.0	2	158	123	281	59.3	0.0
0.17	0.1	2	139	87	226	76.4	0.0
0.17	0.5	2	139	90	229	71.9	0.0
0.17	1.0	2	139	5	144	78.5	0.0
0.33	0.1	2	148	124	272	132.3	0.0
0.33	0.5	2	148	104	252	126.2	0.0
0.33	1.0	2	148	104	252	124.1	0.0

```
NUMBER OF ROWS     NUMBER OF COLUMNS     NUMBER OF UPPER BOUNDS
--------------     -----------------     ----------------------

      50                  300                       0

          REDUCED    OPT.           ITERATIONS             CPU       PERCENT
           COST      SOLN.   -------------------------   TIME     DEVIATION
DENSITY  PERTURB.    TYPE    PHASE 1   PHASE 2   TOTAL   (SEC)  FROM OPTIMUM
------------------------------------------------------------------------------
  0.08      0.1       1        156       161      317     82.2       0.0
  0.08      0.5       1        156       142      298     77.5       0.0
  0.08      1.0       1        156       139      295     76.1       0.0
  0.17      0.1       1        179       130      309    116.4       0.0
  0.17      0.5       1        179       130      309    115.4       0.0
  0.17      1.0       1        179       130      309    118.7       0.0
  0.33      0.1       1        176       134      310    161.4       0.0
  0.33      0.5       1        176       118      294    151.6       0.0
  0.33      1.0       1        176       120      296    144.0       0.0
  0.08      0.1       0        239       219      458    100.6       0.0
  0.08      0.5       0        240       220      460     98.3       0.0
  0.08      1.0       0        244       243      487    113.8       0.0
  0.17      0.1       0        237       209      446    152.3       0.0
  0.17      0.5       0        237       257      494    171.2       0.0
  0.17      1.0       0        237       206      443    154.6       0.0
  0.33      0.1       0        169       161      330    176.1       0.0
  0.33      0.5       0        169       139      308    162.9       0.0
  0.33      1.0       0        169       133      302    164.9       0.0
  0.08      0.1       2        181       165      346     87.9       0.0
  0.08      0.5       2        181       169      350     86.5       0.0
  0.08      1.0       2        181       200      381     91.7       0.0
  0.17      0.1       2        179       132      311    113.6       0.0
  0.17      0.5       2        179       113      292    105.6       0.0
  0.17      1.0       2        179       113      292    103.7       0.0
  0.33      0.1       2        113       144      257    146.6       0.0
  0.33      0.5       2        113       144      257    140.9       0.0
  0.33      1.0       2        113       138      251    140.0       0.0
```

```
NUMBER OF ROWS     NUMBER OF COLUMNS     NUMBER OF UPPER BOUNDS
--------------     -----------------     ----------------------

      50                  300                       25

          REDUCED    OPT.           ITERATIONS             CPU       PERCENT
           COST      SOLN.   -------------------------   TIME     DEVIATION
DENSITY  PERTURB.    TYPE    PHASE 1   PHASE 2   TOTAL   (SEC)  FROM OPTIMUM
------------------------------------------------------------------------------
  0.08      0.1       1        193       205      398    101.9       0.0
  0.08      0.5       1        193       225      418    101.6       0.0
  0.08      1.0       1        193       179      372     91.6       0.0
  0.17      0.1       1        159       185      344    133.1       0.0
  0.17      0.5       1        159       175      334    126.7       0.0
  0.17      1.0       1        159       179      338    131.8       0.0
  0.33      0.1       1        162       153      315    166.7       0.0
  0.33      0.5       1        162       156      318    170.9       0.0
  0.33      1.0       1        162       143      305    162.2       0.0
  0.08      0.1       0        212       225      437    104.4       0.0
  0.08      0.5       0        212       166      378     66.6       0.0
  0.08      1.0       0        212       157      369     97.2       0.0
  0.17      0.1       0        201       193      394    137.1       0.0
  0.17      0.5       0        201       158      359    117.8       0.0
  0.17      1.0       0        201       160      361    125.7       0.0
  0.34      0.1       0        157       167      324    169.2       0.0
  0.34      0.5       0        157       156      313    173.5       0.0
  0.34      1.0       0        157       167      324    177.1       0.0
  0.08      0.1       2        201       151      352     83.4       0.0
  0.08      0.5       2        201       150      351     81.8       0.0
  0.08      1.0       2        201       161      362     87.1       0.0
  0.17      0.1       2        194       115      309    116.9       0.0
  0.17      0.5       2        194       133      327    117.4       0.0
  0.17      1.0       2        194       127      321    109.0       0.0
  0.33      0.1       2        120       157      277    151.4       0.0
  0.33      0.5       2        120       142      262    151.3       0.0
  0.33      1.0       2        120       144      264    153.0       0.0
```

NUMBER OF ROWS	NUMBER OF COLUMNS	NUMBER OF UPPER BOUNDS
100	100	0

DENSITY	REDUCED COST PERTURB.	CPT. SOLN. TYPE	ITERATIONS			CPU TIME (SEC)	PERCENT DEVIATION FROM OPTIMUM
			PHASE 1	PHASE 2	TOTAL		
0.08	0.1	1	165	0	165	59.3	0.0
0.08	0.5	1	165	0	165	65.1	0.0
0.08	1.0	1	165	0	165	64.6	0.0
0.17	0.1	1	180	0	180	93.3	0.0
0.17	0.5	1	180	0	180	94.9	0.0
0.17	1.0	1	180	0	180	93.5	0.0
0.33	0.1	1	187	0	187	112.7	0.0
0.33	0.5	1	187	0	187	143.5	0.0
0.33	1.0	1	187	0	187	148.1	0.0
0.08	0.1	0	165	8	173	59.6	0.0
0.08	0.5	0	166	13	179	61.9	0.0
0.08	1.0	0	165	7	172	59.6	0.0
0.17	0.1	0	186	6	192	103.5	0.0
0.17	0.5	0	186	7	193	120.1	0.0
0.17	1.0	0	186	7	193	105.4	0.0
0.33	0.1	0	176	0	176	139.9	0.0 *
0.33	0.5	0	175	0	175	129.5	0.0 *
0.33	0.5	0	176	0	176	140.4	0.0 * +
0.33	1.0	0	175	0	175	128.6	0.0 *
0.33	1.0	0	176	0	176	136.2	0.0 * +
0.08	0.1	2	181	5	186	55.5	1.80
0.08	0.1	2	164	0	164	59.7	0.0 +
0.08	0.5	2	164	0	164	60.8	0.0
0.08	1.0	2	164	0	164	59.4	0.0
0.17	0.1	2	179	0	179	112.6	0.0
0.17	0.5	2	179	0	179	92.8	0.0
0.17	1.0	2	179	0	179	117.1	0.0
0.33	0.1	2	163	0	163	142.9	0.0
0.33	0.5	2	163	0	163	146.8	0.0
0.33	1.0	2	163	0	163	146.8	0.0

NUMBER OF ROWS	NUMBER OF COLUMNS	NUMBER OF UPPER BOUNDS
100	100	50

DENSITY	REDUCED COST PERTURB.	CPT. SOLN. TYPE	ITERATIONS			CPU TIME (SEC)	PERCENT DEVIATION FROM OPTIMUM
			PHASE 1	PHASE 2	TOTAL		
0.08	0.1	1	147	8	155	60.3	0.0
0.08	0.5	1	147	9	156	58.7	0.0
0.08	1.0	1	147	13	160	65.7	0.0
0.17	0.1	1	176	21	197	114.0	0.0
0.17	0.5	1	177	7	184	96.9	0.0
0.17	1.0	1	177	5	182	100.8	0.0
0.33	0.1	1	176	7	183	136.2	0.0
0.33	0.5	1	177	3	180	118.0	0.0
0.33	1.0	1	176	9	185	166.1	0.0
0.08	0.1	0	149	3	152	59.1	0.0
0.08	0.1	0	147	8	155	58.6	0.0 +
0.08	0.5	0	147	14	161	56.6	0.0
0.08	1.0	0	147	7	154	52.3	0.0
0.17	0.1	0	162	9	171	92.6	0.0
0.17	0.5	0	162	13	175	107.7	0.0
0.17	1.0	0	162	12	174	90.2	0.0
0.33	0.1	0	163	8	171	125.2	0.0
0.33	0.5	0	163	12	175	125.1	0.0
0.33	1.0	0	163	9	172	119.2	0.0

NUMBER OF ROWS	NUMBER OF COLUMNS	NUMBER OF UPPER BOUNDS
100	200	0

DENSITY	REDUCED COST PERTURB.	CPT. SOLN. TYPE	ITERATIONS			CPU TIME (SEC)	PERCENT DEVIATION FROM OPTIMUM
			PHASE 1	PHASE 2	TOTAL		
0.04	0.1	0	436	142	578	225.4	0.0
0.04	1.0	0	436	130	566	217.7	0.0
0.08	0.1	0	468	156	624	468.3	0.0
0.08	1.0	0	468	178	646	471.7	0.0
0.04	0.1	2	361	64	425	165.8	0.05
0.04	1.0	2	361	27	388	152.2	1.85
0.04	1.0	2	361	65	426	162.1	0.48 +
0.08	0.1	2	397	70	467	363.1	0.08
0.08	0.1	2	397	91	488	382.1	0.0 +
0.08	1.0	2	397	85	482	377.4	0.0

NUMBER OF ROWS	NUMBER OF COLUMNS	NUMBER OF UPPER BOUNDS
100	200	50

DENSITY	REDUCED COST PERTURB.	OPT. SOLN. TYPE	ITERATIONS PHASE 1	ITERATIONS PHASE 2	ITERATIONS TOTAL	CPU TIME (SEC)	PERCENT DEVIATION FROM OPTIMUM	
0.04	0.1	0	385	204	589	223.5	0.0	
0.04	0.1	0	385	9	394	145.8	0.52	+
0.04	1.0	0	385	182	567	225.8	0.0	
0.08	0.1	0	421	143	564	403.4	0.0	
0.08	1.0	0	421	63	484	349.7	1.02	
0.08	1.0	0	421	41	462	323.5	1.60	+
0.04	0.1	2	285	71	356	154.6	0.09	
0.04	0.1	2	285	129	414	176.0	0.0	+
0.04	1.0	2	285	133	418	169.5	0.0	
0.08	0.1	2	382	150	532	376.6	0.0	
0.08	1.0	2	382	86	468	331.6	0.93	
0.08	1.0	2	382	181	563	420.0	0.0	+

NUMBER OF ROWS	NUMBER OF COLUMNS	NUMBER OF UPPER BOUNDS
100	300	0

DENSITY	REDUCED COST PERTURB.	OPT. SOLN. TYPE	ITERATIONS PHASE 1	ITERATIONS PHASE 2	ITERATIONS TOTAL	CPU TIME (SEC)	PERCENT DEVIATION FROM OPTIMUM	
0.04	0.1	0	572	423	995	402.1	0.0	
0.04	1.0	0	606	171	777	340.3	2.90	
0.04	1.0	0	573	250	823	342.5	1.41	+
0.04	0.1	2	473	105	578	292.2	0.73	
0.04	0.1	2	473	108	581	294.0	0.70	+
0.04	1.0	2	473	248	721	305.5	0.0	

NUMBER OF ROWS	NUMBER OF COLUMNS	NUMBER OF UPPER BOUNDS
100	300	50

DENSITY	REDUCED COST PERTURB.	OPT. SOLN. TYPE	ITERATIONS PHASE 1	ITERATIONS PHASE 2	ITERATIONS TOTAL	CPU TIME (SEC)	PERCENT DEVIATION FROM OPTIMUM	
0.04	0.1	0	579	153	732	288.2	0.20	
0.04	0.1	0	579	88	667	270.5	0.32	+
0.04	1.0	0	579	415	994	356.9	0.0	
0.04	0.1	2	376	295	671	308.2	0.0	
0.04	1.0	2	376	228	604	262.8	0.0	

NUMBER OF ROWS	NUMBER OF COLUMNS	NUMBER OF UPPER BOUNDS
150	300	0

DENSITY	REDUCED COST PERTURB.	OPT. SOLN. TYPE	ITERATIONS			CPU TIME (SEC)	PERCENT DEVIATION FROM OPTIMUM	
			PHASE 1	PHASE 2	TOTAL			
0.04	0.1	0	1663	32	1695	1259.2	0.0	
0.04	1.0	0	1774	60	1834	1322.9	0.0	
0.04	0.1	0	927	31	958	869.2	0.53	
0.04	0.1	0	954	8	962	887.7	0.52	+
0.04	1.0	0	947	68	1015	927.9	3.74	
0.04	1.0	0	979	13	992	925.2	6.81	+

NUMBER OF ROWS	NUMBER OF COLUMNS	NUMBER OF UPPER BOUNDS
150	300	50

DENSITY	REDUCED COST PERTURB.	OPT. SOLN. TYPE	ITERATIONS			CPU TIME (SEC)	PERCENT DEVIATION FROM OPTIMUM	
			PHASE 1	PHASE 2	TOTAL			
0.04	0.1	0	1430	88	1518	1587.7	0.05	
0.04	1.0	0	1393	6	1399	1455.7	1.05	
0.04	0.1	0	787	198	985	1187.4	0.07	
0.04	0.1	0	787	105	892	1054.6	0.0	†
0.04	1.0	0	787	91	878	1006.7	1.79	

for the optimal solution type

0	indicates degenerate
1	indicates unique
2	indicates alternative

SENSITIVITY ANALYSIS FOR PARAMETRIC NONLINEAR PROGRAMMING USING PENALTY METHODS

Robert L. Armacost
U.S. Coast Guard Headquarters

Anthony V. Fiacco
Department of Operations Research
The George Washington University

Abstract

Recently, it has been shown that a class of penalty function algorithms can readily be adapted to generate sensitivity analysis information for a large class of parametric nonlinear programming problems. In particular, estimates of the partial derivatives (with respect to the problem parameters) of the components of a solution vector and the optimal value function have been successfully calculated for a number of nontrivial examples. The approach has been implemented using the well-known Sequential Unconstrained Minimization Technique (SUMT) computer program. This paper briefly summarizes these results, presents additions to the computer program that include a screening device for eliminating calculations associated with less important parameters, and illustrates the kind of information that can be generated by applying the technique to a well-known inventory model.

1. Introduction

Initial numerical results resulting from the implementation of a penalty function technique for obtaining sensitivity information in parametric nonlinear programming were given by Armacost and Fiacco (1974). The work is based on the theory developed by Fiacco and McCormick (1968) and extended by Fiacco (1973). This paper reports on refinements and extensions of the computational procedures implemented by Armacost and Mylander (1973) and Armacost (1976), using the SUMT-Version 4 computer code with the logarithmic-quadratic loss penalty function to estimate the partial derivatives of the solution point and the objective function optimal value, the derivatives here being taken with respect to the specified problem parameters.

Fiacco (1973) developed the necessary general formulas for the partial derivatives of the "optimal value function," the components of a local solution point and its associated optimal Lagrange multipliers, for a large class of parametric nonlinear programming problems composed of twice differentiable functions. He also obtained approximation formulas in terms of the well-known logarithmic-quadratic penalty function. Recently, Armacost and Fiacco (1975) particularized and simplified these formulas for various problem structures and developed formulas for the first and second derivatives of the optimal value function of the given problem. Additionally, Armacost and Fiacco (1976) have applied the general theory to easily prove the well-known result that, when the parameters are the right-hand side components of the constraints, the optimal Lagrange multipliers give the gradient of the optimal value function (with respect to the parameters). Further, it was shown that the first derivatives of the Lagrange multipliers give the components of the Hessian of the optimal value function, and explicit formulas were developed for the Hessian in terms of the problem functions.

In their first report on computational experience, Armacost and Fiacco (1974) concentrated primarily on presenting computational experience associated with the calculation of the first derivatives of a local solution point. The practical implementability of the approach was demonstrated.

In a subsequent paper, Armacost (1976) reported on additional computational experience, focusing on the calculation of the derivatives of the optimal value function and the Lagrange multipliers, also implementing a potentially valuable refinement that allows for computerized screening for "key" parameters. This paper may be regarded as a continuation and amplification of the Armacost paper.

For problems involving a large number of parameters, a very large number of partial derivatives may be calculated if one proceeds indiscriminately. This is not only time-consuming, but may also be quite burdensome to a user who must evaluate the overall significance of the results. One measure of the latter is the effect of a perturbation on the solution value. It is quite possible and often observed in practice that the optimal objective function value is much more sensitive to a few of the many parameters present. With this in mind, the method developed by Armacost and Fiacco (1975) to estimate the first order sensitivity of the optimal value function was incorporated in the computer program to provide an option for preliminary screening of the parameters to eliminate further calculations involving perturbations of parameters having "little" effect on the optimal value function. (A user can easily introduce his own criteria of significance in this determination.) Using the formulas developed by Fiacco (1973), a second option is included which permits the calculation of the sensitivity estimates for the Lagrange multipliers.

In Section 3, a sensitivity analysis is conducted for a multi-item inventory model developed by Schrady and Choe (1971) for the U.S. Navy. The example analyzed is the same small one treated by Schrady and Choe, though readily extended to a large-scale model. The results illustrate the potential value of a detailed automated sensitivity analysis in practical situations, and hopefully dramatize the numerous rich interpretations and insights that can be derived from this information, as well as indicating the caution that must be taken in making valid inferences.

The recently obtained basic theoretical results validating the computational algorithm are summarized rather completely in the next section so that the paper might be self-contained.

2. Supporting Theory

The parametric mathematical programming problems considered here are of the form

$$\begin{aligned} &\underset{x \in E^n}{\text{minimize}} \quad f(x,\epsilon) \\ &\text{subject to} \quad g_i(x,\epsilon) \geq 0 \ , \quad i=1,\ldots,m \ , \qquad P(\epsilon) \\ &\qquad\qquad\quad h_j(x,\epsilon) = 0 \ , \quad j=1,\ldots,p \ , \end{aligned}$$

where x is the usual vector of variables and ϵ is a k-component vector of numbers called "parameters." It is desired ultimately to develop a complete characterization of a solution $x(\epsilon)$ of Problem $P(\epsilon)$ as a function of ϵ. In our current work, we have concentrated on certain recently computationally tractable measures of change in a solution as ϵ is perturbed from a specified value. (Without loss of generality, we assume that the specified value is $\epsilon = 0$.)

When certain assumptions are satisfied, Fiacco (1973) and Armacost and Fiacco (1975) have characterized the "first order sensitivity" of a "Kuhn-Tucker Triple" and the first and second order sensitivity of the optimal value function of Problem $P(\epsilon)$. (These quantities are defined as the theory is presented.) Additionally, they have developed formulas for efficiently estimating this sensitivity when the logarithmic-quadratic loss penalty function algorithm is used to solve Problem $P(\epsilon)$. The main theoretical results are summarized here.

The Lagrangian for Problem $P(\epsilon)$ is defined as

$$L(x,u,w,\epsilon) \equiv f(x,\epsilon) - \sum_{i=1}^{m} u_i g_i(x,\epsilon) + \sum_{j=1}^{p} w_j h_j(x,\epsilon) \ ,$$

where u_i, $i=1,\ldots,m$ and w_j, $j=1,\ldots,p$ are "Lagrange multipliers" associated with the inequality and equality constraints, respectively. Any vector $(\bar{x},\bar{u},\bar{w})$ satisfying the usual (first order) Kuhn-Tucker conditions (Fiacco and McCormick, 1968) of Problem $P(\bar{\epsilon})$ is called a Kuhn-Tucker triple.

The following four assumptions are sufficient to establish the desired results and are assumed to hold throughout the paper:

A1 — The functions defining Problem $P(\epsilon)$ are twice continuously differentiable in (x,ϵ) in a neighborhood of $(x^*,0)$.

A2 — The second order sufficient conditions for a local minimum of Problem $P(0)$ hold at x^* with associated Lagrange multipliers u^* and w^*.

A3 — The gradients $\nabla_x g_i(x^*,0)$ (i.e., $(\partial g_i(x^*,0)/\partial x_1,\ldots,\partial g_i(x^*,0)/\partial x_n)^T$, the superscript T denoting transportation) for all i such that $g_i(x^*,0) = 0$, and $\nabla_x h_j(x^*,0)$, $j=1,\ldots,p$ are linearly independent.

A4 — Strict complementary slackness holds at $(x^*,0)$ (i.e., $u_i^* > 0$ for all i such that $g_i(x^*,0) = 0$).

Theorem 1: (Local characterization of a Kuhn-Tucker Triple (Fiacco, 1973) of Problem $P(\epsilon)$.) If assumptions A1, A2, A3 and A4 hold for Problem $P(\epsilon)$ at $(x^*,0)$, then

(a) x^* is a local isolated minimizing point of Problem $P(0)$ and the associated Lagrange multipliers u^* and w^* are unique;

(b) for ϵ in a neighborhood of 0, there exists a unique, once continuously differentiable vector function $y(\epsilon) = (x(\epsilon),u(\epsilon),w(\epsilon))^T$ satisfying the second order sufficient conditions for a local minimum of Problem $P(\epsilon)$ such that $y(0) = (x^*,u^*,w^*)^T = y^*$ and hence, $x(\epsilon)$ is a locally unique, local minimum of Problem $P(\epsilon)$ with associated unique Lagrange multipliers $u(\epsilon)$ and $w(\epsilon)$; and

(c) for ϵ near 0, the set of binding inequalities is unchanged, strict complementary slackness holds for $u_i(\epsilon)$ for i such that $g_i(x(\epsilon),\epsilon) = 0$, and the binding constraint gradients are linearly independent at $x(\epsilon)$.

This result provides a characterization of a local solution of Problem $P(\epsilon)$ and its associated optimal Lagrange multipliers near $\epsilon = 0$. It generalizes a theorem first presented by Fiacco and McCormick (1968, Theorem 6) and is closely related to a generalization of the same theorem provided independently by Robinson (1974). It shows that the Kuhn-Tucker triple $y(\epsilon)$ is unique and well behaved, under the given conditions. Since $y(\epsilon)$ is once differentiable, the partial derivatives of the components of $y(\epsilon)$ are well defined. This fact and Assumption A1 also mean that the functions defining Problem $P(\epsilon)$ are once continuously differentiable functions of ϵ along the "solution trajectory" $x(\epsilon)$ near $\epsilon = 0$, and the Lagrangian is a once continuously differentiable function of ϵ along the "Kuhn-Tucker point trajectory."

We are thus motivated to determine a means to calculate the various partial derivatives, since this yields a first order estimate of the locally optimal Kuhn-Tucker triple and the problem functions near $\varepsilon = 0$.

Denote by $\nabla_\varepsilon x(\varepsilon) \equiv (\partial x_i(\varepsilon)/\partial\varepsilon_j)$, $i=1,\dots,n$, $j=1,\dots,k$, the $n \times k$ matrix of partial derivatives of $x(\varepsilon)$ with respect to ε , and define $\nabla_\varepsilon u(\varepsilon)$ and $\nabla_\varepsilon w(\varepsilon)$ in a similar fashion. We then define $\nabla_\varepsilon y(\varepsilon) \equiv (\nabla_\varepsilon^T x(\varepsilon), \nabla_\varepsilon^T u(\varepsilon), \nabla_\varepsilon^T w(\varepsilon))^T$, an $(n+m+p) \times k$ matrix.

When $y(\varepsilon)$ is available, $\nabla_\varepsilon y(\varepsilon)$ can be calculated by noting that Conclusion (b) of the theorem implies the satisfaction of the Kuhn-Tucker conditions for $P(\varepsilon)$ at $y(\varepsilon)$ near $\varepsilon = 0$, i.e.,

$$\begin{aligned} &\nabla_x L[x(\varepsilon),u(\varepsilon),w(\varepsilon),\varepsilon] = 0 \ , \\ &u_i(\varepsilon)g_i[x(\varepsilon),\varepsilon] = 0 \ , \quad i=1,\dots,m \ , \\ &h_j[x(\varepsilon),\varepsilon] = 0 \ , \quad j=1,\dots,p \ . \end{aligned} \tag{1}$$

Since the Jacobian $M(\varepsilon)$ of this system with respect to (x,u,w) (i.e., the matrix obtained by differentiating the left side of (1) with respect to the components of (x,u,w)) is nonsingular under the given assumptions, the total derivative of the system with respect to ε is well defined and must equal zero. This yields

$$M(\varepsilon)\nabla_\varepsilon y(\varepsilon) = N(\varepsilon) \ ,$$

where $N(\varepsilon)$ is the negative of the Jacobian of the Kuhn-Tucker system with respect to ε , and hence

$$\nabla_\varepsilon y(\varepsilon) = M(\varepsilon)^{-1}N(\varepsilon) \ .$$

The class of algorithms based on twice continuously differentiable penalty functions can be used without additional assumptions and without requiring $y(\varepsilon)$ to provide an estimate of $\nabla_\varepsilon y(\varepsilon)$. Furthermore, most of the information required to make the estimate is already available in the typical implementations of these algorithms. Here, we use the logarithmic-quadratic penalty function for Problem $P(\varepsilon)$ (Fiacco and McCormick, 1968) defined as

$$W(x,\varepsilon,r) \equiv f(x,\varepsilon) - r \sum_{i=1}^{m} \ln g_i(x,\varepsilon) + (1/2r) \sum_{j=1}^{p} h_j^2(x,\varepsilon) \ . \tag{2}$$

Under the given assumptions, the following facts are known for Problem $P(0)$ from penalty function theory (Fiacco and McCormick, 1968, Theorems 10 and 17):

(1) For $r > 0$ and small, there exists a unique once continuously differentiable vector function $x(0.r)$ such that $x(0,r)$ is a locally unique minimizing point of $W(x,0,r)$ in $R^{\ddagger}(0) \equiv \{x: g_i(x,0) > 0 \ , \ i=1,\dots,m \ ,$ and $h_j(x,0) = 0 \ , \ j=1,\dots,p\}$ and such that

$$x(0,r) \to x(0,0) = x^* \ ;$$

(2) $\lim\limits_{r\to 0} r \sum\limits_{i=1}^{m} \ln g_i[x(0,r)] = 0$;

(3) $\lim\limits_{r\to 0} (1/2r) \sum\limits_{j=1}^{p} h_j^2[x(0,r),0] = 0$; and

(4) $\lim\limits_{r\to 0} W[x(0,r),0,r] = f(x^*,0)$.

The following theorem extends these results for Problem $P(\varepsilon)$, where ε is allowed to vary in a neighborhood of 0 , and provides a basis for approximating the sensitivity information associated with Problem $P(\varepsilon)$. The notation $\nabla_x^2 W$ denotes the matrix of second partial derivatives of W with respect to x .

Theorem 2: (Relationship of solutions of Problem $P(\varepsilon)$ and minima of $W(x,\varepsilon,r)$, (Fiacco 1973). If Assumptions A1 - A4 hold, then in a neighborhood about $(\varepsilon,r) = (0,0)$ there exists a unique once continuously differentiable vector function $y(\varepsilon,r) = [x(\varepsilon,r),u(\varepsilon,r),w(\varepsilon,r)]^T$ satisfying

$$\begin{aligned} &\nabla_x L(x,u,w,\varepsilon) = 0 \ , \\ &u_i g_i(x,\varepsilon) = r \qquad i=1,\dots,m \ , \\ &h_j(x,\varepsilon) = w_j r \ , \quad j=1,\dots,p \ , \end{aligned}$$

with $y(0,0) = (x^*,u^*,w^*)$ and such that, for any (ε,r) near $(0,0)$ and $r > 0$, $x(\varepsilon,r)$ is a locally unique unconstrained local minimizing point of $W(x,\varepsilon,r)$, $g_i[x(\varepsilon,r),\varepsilon] > 0$, $i=1,\dots,m$, and $\nabla_x^2 W[x(\varepsilon,r),\varepsilon,r]$ is positive definite.

Corollary 2.1: (Convergence of estimates using $W(x,\varepsilon,r)$, (Fiacco, 1973).) If Assumptions A1, A2, A3 and A4 hold for Problem $P(\varepsilon)$, then for any ε near 0 ,

(a) $\lim\limits_{r\to 0^+} y(\varepsilon,r) = y(\varepsilon,0) = y(\varepsilon)$, the Kuhn-Tucker triple characterized in Theorem 1; and

(b) $\lim\limits_{r\to 0^+} \nabla_\varepsilon y(\varepsilon,r) = \nabla_\varepsilon y(\varepsilon,0) = \nabla_\varepsilon y(\varepsilon)$.

This result motivates use of $\nabla_\varepsilon y(\varepsilon,r)$ to estimate $\nabla_\varepsilon y(\varepsilon)$, when ε is near 0 and r is near 0 , once $y(\varepsilon,r)$ is available. Theorem 2 provides the basis for an efficient calculation of $\nabla_\varepsilon y(\varepsilon,r)$. Since, at a local solution point $x(\varepsilon,r)$ of $W(x,\varepsilon,r)$, it follows that

$$\nabla_x W[x(\varepsilon,r),\varepsilon,r] = 0 \ , \tag{3}$$

we can differentiate (3) with respect to ε to obtain

$$\nabla_x^2 W[x(\varepsilon,r),\varepsilon,r]\nabla_\varepsilon x(\varepsilon,r) + \nabla_\varepsilon(\nabla_x W[x(\varepsilon,r),\varepsilon,r]) = 0 \;. \qquad (4)$$

By Theorem 2, $\nabla_x^2 W$ is positive definite for (ε,r) near $(0,0)$ and $r > 0$, so $\nabla_x^2 W$ has an inverse and $\nabla_\varepsilon x(\varepsilon,r) = -\nabla_x^2 W[x(\varepsilon,r),\varepsilon,r]^{-1} \cdot \nabla_{\varepsilon x}^2 W[x(\varepsilon,r),\varepsilon,r]$.

Also, since

$$u_i(\varepsilon,r) = r/g_i(x(\varepsilon,r),\varepsilon) \;, \quad i=1,\ldots,m \;, \qquad (5)$$

and

$$w_j(\varepsilon,r) = h_j(x(\varepsilon,r),\varepsilon)/r \;, \quad j=1,\ldots,p \;, \qquad (6)$$

for (ε,r) near $(0,0)$ and $r > 0$, these equations can be differentiated with respect to ε to obtain

$$\nabla_\varepsilon u_i(\varepsilon,r) = -(r/g_i^2)[\nabla_x g_i(x(\varepsilon,r),\varepsilon) \cdot \nabla_\varepsilon x(\varepsilon,r) + \partial g_i(x(\varepsilon,r),\varepsilon)/\partial\varepsilon] \;, \qquad (7)$$

$$\nabla_\varepsilon w_j(\varepsilon,r) - (1/r)[\nabla_x h_j(x(\varepsilon,r),\varepsilon) \cdot \nabla_\varepsilon x(\varepsilon,r) + \partial h_j(x(\varepsilon,r),\varepsilon)/\partial\varepsilon] \;. \qquad (8)$$

Solving (4) and calculating (7) and (8) then yields the components of $\nabla_\varepsilon y(\varepsilon,r)$, which can be used to estimate $\nabla_\varepsilon y(\varepsilon)$ for (ε,r) near $(0,0)$.

The next results extend this theory to an analysis of the optimal value function of Problem $P(\varepsilon)$ along the Kuhn-Tucker point trajectory $[x(\varepsilon),u(\varepsilon),w(\varepsilon)]^T$.

The optimal value function is defined as:

$$f^*(\varepsilon) \equiv f[x(\varepsilon),\varepsilon] \;, \qquad (9)$$

and the "optimal value Lagrangian" is defined as:

$$L^*(\varepsilon) = L[x(\varepsilon),u(\varepsilon),w(\varepsilon),\varepsilon] \;. \qquad (10)$$

Theorem 3: (First and second order changes in the optimal value function, Armacost and Fiacco (1975).) If assumptions A1 - A4 hold for Problem $P(\varepsilon)$, then for ε near 0, $f^*(\varepsilon)$ is a twice continuously differentiable function of ε, and

(a) $f^*(\varepsilon) = L^*(\varepsilon)$;

(b) $$\nabla_\varepsilon f^*(\varepsilon) = \nabla_\varepsilon L(x,u,w,\varepsilon)\Big|_{(x,u,w)=(x(\varepsilon),u(\varepsilon),w(\varepsilon))} = \nabla_\varepsilon f(x,\varepsilon) - \sum_{i=1}^{m} u_i \nabla_\varepsilon g_i(x,\varepsilon) + \sum_{j=1}^{p} w_j \nabla_\varepsilon h_j(x,\varepsilon)\Big|_{(x,u,w)=(x(\varepsilon),u(\varepsilon),w(\varepsilon))} \;;$$

(c) $\nabla_\varepsilon^2 f^*(\varepsilon) = \nabla_\varepsilon(\nabla_\varepsilon L(x(\varepsilon),u(\varepsilon),w(\varepsilon),\varepsilon)^T)$.

The logarithmic-quadratic loss penalty function (2) can also be used to provide estimates of the first and second order sensitivity of the optimal value function. Let the optimal value penalty function be defines as $W^*(\varepsilon,r) = W(x(\varepsilon,r),\varepsilon,r)$.

Theorem 4: (First and second order sensitivity of $W^*(\varepsilon,r)$ and estimates for $f^*(\varepsilon)$, Armacost and Fiacco (1975).) If Assumptions A1 - A4 hold for Problem $P(\varepsilon)$, then for (ε,r) near $(0,0)$ and $r > 0$, $W^*(\varepsilon,r)$ is a twice continuously differentiable function of ε and

(a) $\lim_{r\to 0^+} W^*(\varepsilon,r) = L^*(\varepsilon) = f^*(\varepsilon)$;

(b) $$\nabla_\varepsilon W^*(\varepsilon,r) = \nabla_x W \, \nabla_\varepsilon X + \nabla_\varepsilon W = \nabla_\varepsilon L(x,u,w,\varepsilon)\Big|_{(x,u,w)=(x(\varepsilon,r),u(\varepsilon,r),w(\varepsilon,r))} \;;$$

(c) $$\lim_{r\to 0^+} \nabla_\varepsilon W^*(\varepsilon,r) = \nabla_\varepsilon L(x(\varepsilon),u(\varepsilon),w(\varepsilon),\varepsilon) = \nabla_\varepsilon f^*(\varepsilon) \;; \qquad (11)$$

(d) $\nabla_\varepsilon^2 W^*(\varepsilon,r) = \nabla_\varepsilon(\nabla_\varepsilon L(x(\varepsilon,r),u(\varepsilon,r),w(\varepsilon,r),\varepsilon)^T)$;

(e) $\lim_{r\to 0^+} \nabla_\varepsilon^2 W^*(\varepsilon,r) = \nabla_\varepsilon^2 f^*(\varepsilon)$.

This result provides a justification for estimating $f^*(\varepsilon)$, $\nabla_\varepsilon f^*(\varepsilon)$ and $\nabla_\varepsilon^2 f^*(\varepsilon)$ by $W^*(\varepsilon,r)$, $\nabla_\varepsilon W^*(\varepsilon,r)$ and $\nabla_\varepsilon^2 W^*(\varepsilon,r)$, respectively, when r is positive and small enough.

Since Corollary 2.1 and continuity imply that $\lim_{r\to 0^+} f(x(\varepsilon,r),\varepsilon) = f^*(\varepsilon)$, another estimate of the optimal value function (9) is provided by $f^\#(\varepsilon,r) \equiv f(x(\varepsilon,r),\varepsilon)$ when $r > 0$ and small. Direct application of the chain rule for differentiation then yields, for $x = x(\varepsilon,r)$,

$$\nabla_\varepsilon f^\#(\varepsilon,r) = \nabla_x f(x,\varepsilon)\nabla_\varepsilon x(\varepsilon,r) + \nabla_\varepsilon f(x,\varepsilon) . \qquad (12)$$

Under the given assumptions, continuity also assures that $\nabla_\varepsilon f^\#(\varepsilon,r) \to \nabla_\varepsilon f^*(\varepsilon)$ as $r \to 0^+$. Thus both

$\nabla_\varepsilon f^{\#}(\varepsilon,r)$ and $\nabla_\varepsilon W^*(\varepsilon,r)$ are estimates of $\nabla_\varepsilon f^*(\varepsilon)$ for r sufficiently small.

It should be noted that these estimates are functionally related since

$$\nabla_\varepsilon W^*(\varepsilon,r) = \nabla_\varepsilon f^{\#}(\varepsilon,r) - \sum_{i=1}^{m} u_i(\nabla_x g_i \nabla_\varepsilon x(\varepsilon,r) + \nabla_\varepsilon g_i) + \sum_{j=1}^{p} w_j(\nabla_x h_j \nabla_\varepsilon x(\varepsilon,r) + \nabla_\varepsilon h_j) \Big|_{x = x(\varepsilon,r)} .$$

From this expression, it is clear that $\nabla_\varepsilon f^{\#}(\varepsilon,r)$ is the better estimate of $\nabla_\varepsilon f^*(\varepsilon)$, the remaining terms in $\nabla_\varepsilon W^*(\varepsilon,r)$ simply constituting "noise" that is eliminated as $r \to 0^+$. However, by using the expression for $\nabla_\varepsilon W^*(\varepsilon,r)$ given by (11), $\nabla_\varepsilon W^*(\varepsilon,r)$ can be evaluated without necessitating the calculation of $\nabla_\varepsilon x(\varepsilon,r)$, which is required to compute (12). Thus, the cruder but computationally much cheaper estimate of $\nabla_\varepsilon f^*(\varepsilon)$ given by Equation (11) has now been introduced as an option in the computer program as a preliminary screening device to identify crucial parameters. Restriction of subsequent calculations to these parameters, and other calculations such as the sharper estimate of $\nabla f^*(\varepsilon)$ given by (12) are provided as additional options.

In summary, the basis for the estimation procedure utilized here for a specific problem, say Problem $P(\varepsilon)$, is the minimization of the penalty function $W(x,\varepsilon,r)$ given by (2). This yields a point $x(\varepsilon,r)$ which may be viewed as an estimate of a (local) solution $x(\varepsilon)$ of Problem $P(\varepsilon)$. The estimate $f(x(\varepsilon,r),\varepsilon)$ of $f^*(\varepsilon)$ is immediately available when $x(\varepsilon,r)$ has been determined. The associated optimal Lagrange multipliers $u(\varepsilon)$ and $w(\varepsilon)$ are estimated by using the relationships given in (5) and (6), respectively.

If desired, all of the first partial derivatives of $y(\varepsilon,r) = (x(\varepsilon,r),u(\varepsilon,r),w(\varepsilon,r)^T$ with respect to ε, an estimate of $\nabla_\varepsilon y(\varepsilon)$. may be obtained by first solving (4) and then applying (7) and (8). If the full matrix $\nabla_\varepsilon x(\varepsilon,r)$ is calculated, then $\nabla_\varepsilon f^*(\varepsilon)$ is estimated by (12). However, if it is desired to eliminate calculations involving parameters having less effect on local changes of $f^*(\varepsilon)$, the screening device described above is used. This entails initial estimation of $\nabla_\varepsilon f^*(\varepsilon)$ by (11). Components of ε corresponding to the respective components of $\nabla_\varepsilon f^*(\varepsilon)$ that are deemed inconsequential, may then be deleted and would not enter into any subsequent calculations. In particular, it is emphasized that (x^*,u^*,w^*), $f^*(\varepsilon)$ and $\nabla_\varepsilon f^*(\varepsilon)$ may be estimated by $y(\varepsilon,r)$ $f(x(\varepsilon,r),\varepsilon)$ and (11b), respectively, without calculating any components of $\nabla_\varepsilon y(\varepsilon,r)$, once $x(\varepsilon,r)$ is known.

3. Example: A Large-scale Multi-item Inventory Model

Traditionally, inventory models have been formulated to minimize some function of the ordering, holding and shortage (or backorder) costs subject to various constraints. Schrady and Choe (1971) have formulated an inventory model which appears to have much greater relevance for an inventory system in a noncommercial environment, such as institutional or military. The costs used in the traditional models may be quite artificial and the real objective of the system is often maximization of a measure of readiness or service, here assumed to be equivalent to minimization of stockouts. In addition, the stock points of such supply systems are inevitably constrained by investment and reorder workload limitations.

Schrady and Choe's multi-item inventory system assumes these constraints along with the specific objective of minimizing the total time-weighted shortages. The decision variables are taken to be the "reorder quantities" and the "reorder points," respectively, how much to order and when to order each item in the inventory. A three-item example problem was solved by Schrady and Choe (1971) using the SUMT computer code (Mylander, et al., 1971). Subsequently, McCormick (1972) showed how the special structure of this inventory model can be used to facilitate the use of the SUMT code to solve very large inventory problems. He also extended the model to include constraints on storage volume and the probability of depletion of critical items.

The model and example presented here are the original ones due to Schrady and Choe. The penalty function technique described in the preceding section was used to solve the example and calculate the partial derivatives of various quantities of interest, with respect to each parameter involved in defining the model. (The analysis can be applied to the extended model without difficulty.)

Detailed development of the model is beyond the scope of this paper. The interested reader is referred to the Schrady-Choe and McCormick papers. Here, we give a summary treatment of the various conditions and relationships upon which the model is based. We then tabulate the results obtained in solving the resulting nonlinear programming problem and applying the sensitivity analysis methodology. A number of observations and interpretations are offered to illustrate the many uses to which the sensitivity information might be applied.

It is assumed that the amount of each item in inventory is always known, that all demand which occurs when the on-hand stock is zero is backordered, and that the demand which occurs during the time between the placement of an order and its receipt by the stock point (i.e., the "lead time demand") is normally distributed with known mean u_i and variance σ_i^2.

For the ith item, let

c_i = item unit cost (in dollars),

λ_i = mean demand per unit time (in units),

r_i = reorder point,

Q_i = reorder quantity,

$\phi(x)$ = the Normal (0,1) density function,

$\Phi(z) = \int_z^\infty \phi(x)dx$ = the Normal (0,1) complementary c lative distribution function.

In addition, let K_1 be the investment limit in dollars, K_2 the number of orders per unit of time that constitutes reorder workload limit, and N the total number of items in the inventory.

It can be shown that the expected time-weighted shortage of item i at any point in time is given by

$$B_i(Q_i,r_i) = \frac{1}{Q_i}\left[\beta_i(r_i) - \beta_i(Q_i+r_i)\right]$$

where

$$\beta_i(r_i) = \frac{1}{2}\left[\sigma_1^2 + (r_i-\mu_i)^2\right]$$

$$\Phi\left(\frac{r_i-\mu_i}{\sigma_i}\right) - \frac{\sigma_i}{2}(r_i-\mu_i)\phi\left(\frac{r_i-\mu_i}{\sigma_i}\right) .$$

The expected on-hand inventory of item i is given by $r_i + Q_i/2 - \mu_i + B_i(Q_i,r_i)$ and the expected number of orders placed per unit time for item i is λ_i/Q_i .

Using the above expressions and assumptions, Schrady and Choe (1971) indicate that meaningful approximations of the given quantities are obtained even when the second term is dropped from the expression for the expected shortages, and when the last term is dropped from the expression for expected on-hand inventory. The given assumptions and simplifications then load readily to the following nonlinear programming problem (which Schrady and Choe (1971) proved convex),

$$\underset{Q,r}{\text{minimize}} \quad Z(Q,r) \equiv \sum_{i=1}^{N} \beta_i(r_i)/Q_i$$

subject to

$$\left.\begin{aligned} g_i(Q,r) &\equiv K_1 - \sum_{i=1}^{N} c_i(r_i+Q_i/2-\mu_i) \geq 0 , \\ g_2(Q,r) &\equiv K_2 - \sum_{i=1}^{N} \lambda_i/Q_i \geq 0 , \end{aligned}\right. \qquad \text{(SC)}$$

with r_i unrestricted in sign, $Q_i \geq 0$, $i=1,\ldots,N$, $Q = (Q_1,\ldots,Q_N)^T$, $r = (r_1,\ldots,r_N)^T$, and g_1 and g_2 representing the investment and workload constraints, respectively.

The problem data for the Schrady-Choe three-item example and the initial starting point for the SUMT program are shown in Table 1. As indicated in the table, the lead-time demands and standard deviations, the item unit costs and mean demands, and the investment and workload limits are all treated as parameters in conducting the sensitivity analysis.

Table 2 gives the computer solution and Table 3 the final estimate of the first partial derivatives of the optimal value function Z^* with respect to the problem parameters. Relative to the criterion used in the computer program, the Table 3 results indicate that the optimal value function is sensitive to parameters K_2 , c_1 , σ_2 , c_2 , σ_3 and c_3 . Many inferences are possible. For example, the fact that the solution is particularly sensitive to the values of the standard deviations of the lead time demand of items 2 and 3 might indicate that, since these parameters were obtained by sampling, additional sampling of these lead time demands may very well be warranted to reduce the associated standard deviations.

Table 3 also suggests that the optimal solution value is very sensitive to all of the item costs. If the structure of Problem (SC) is examined, this result may at first appear contradictory since the c_i appear only in the investment constraint and the optimal value function, according to Table 3, is apparently not very sensitive to the investment limit K_1 . The problem is one of precise interpretation. The partial derivatives measure rate of change. But inspection of the investment constraint $g_1(Q,r)$ at (Q^*,r^*) reveals that the change in an item cost c_i by any amount Δc_i has the same effect on the constraint as a change in the investment limit K_1 of $-(r_i^* + Q_i^*/2 - \mu_i)\Delta c_i$. Since the quantity in parentheses may be verified from Tables 1 and 2 to be much greater than 1 for all i , it follows that the effect of changing any c_i by any increment δ will be much greater on the constraint (and hence, on the optimal value Z^* , since the constraint is binding) than the effect of changing K_1 by the same amount δ . This implies that $|\partial Z^*/\partial c_i| > |\partial Z^*/\partial K_1|$ for each i and, in fact, it can be shown here that $\partial Z^*/\partial c_i = -(r_i^* + Q_i^*/2 - \mu_i)\partial Z^*/\partial K_1$, so that the relationships indicated are indeed precisely verified.

The above observations might also suggest that some care must be taken in interpreting the results. Changes in the parameter associated with the largest (in absolute value) partial derivative will give the greatest local change in the optimal value of the objective function, compared to a change of the same magnitude in any other parameter taken individually. This follows because either the objective function and/or some of the constraints (as above) are most significantly affected by this parameter change at the current solution. General rules have not been given for selection of optimal changes in the parameters, i.e., for determining the optimal magnitude and combination of such changes. It is well beyond the scope of this paper to pursue this "macro-analysis" determination, though it should be noted that the greatest local rate of decrease in the optimal value function is along the direction of the negative of the gradient of this function in parameter space (i.e., along the vector composed of the negative of the components of the partial derivatives with respect to the various parameters). A user would nonetheless have to determine the feasibility of this direction of change and, if feasible, the optimal move along

Table 1
INVENTORY PROBLEM DATA

	QUANTITY	ITEM i 1	2	3	
PARAMETERS	μ_i	100	200	300	(MEAN OF LEAD-TIME DEMAND)
	σ_i	100	100	200	(S.D. OF LEAD-TIME DEMAND)
	c_i	1	10	20	(ITEM UNIT COST - DOLLARS)
	λ_i	1,000	1,500	2,000	(MEAN DEMAND/UNIT TIME)
	K_1	$8,000			(INVESTMENT LIMIT)
	K_2	15 re-orders/unit time			(RE-ORDER WORKLOAD LIMIT)
VAR.	Q_i^0	600	270	300	(AMOUNT ORDERED)
	r_i^0	200	260	400	(RE-ORDER LEVEL)

Table 2
SOLUTION AND LAGRANGE MULTIPLIERS

	QUANTITY	ITEM i 1	2	3
VAR.	Q_i^*	533	246	285
	r_i^*	253	277	437
L.M.	u_1^*	.0552		
	u_2^*	.6230		
VALUE	z^*	12.987		

Table 3
OPTIMAL VALUE FUNCTION DERIVATIVES

PARTIALS	ITEM i 1	2	3
$\partial Z^*/\partial \mu_i$	-.0000	-.0003	-.0008
$\partial Z^*/\partial \sigma_i$	.0119	.0897[a]	.1729[a]
$\partial Z^*/\partial c_i$	2.1713[a]	1.0345[a]	1.4452[a]
$\partial Z^*/\partial \lambda_i$	.0012	.0025	.0022
$\partial Z^*/\partial K_1$	-.0052		
$\partial Z^*/\partial K_2$	-.6230[a]		

[a] Deemed "significant" by criterion, $|\Delta_1 Z^*|/Z^* > .001$ for a unit change in the given parameter, where $\Delta_1 Z^*$ is the estimated first order change in Z^*. This criterion was selected arbitrarily for illustrative purposes. Criteria appropriate to the particular application can be selected by a user.

this vector, taking into account other factors such as the relative "cost-effectiveness" of any schedule of changes in any model parameter.

Referring back to Table 2, we note that the Lagrange multiplier u_2^* is much greater than u_1^* Recalling the "sensitivity" interpretation of Lagrange multipliers, which holds under the present conditions, it follows that $u_1^* = -\partial Z^*/\partial K_1$ and $u_2^* = -\partial Z^*/\partial K_2$. This conclusion is consistent with the result obtained in Table 3, and it means that the workload constraint g_2 is by far the more effective in determining the minimum number of expected time-weighted shortages at the current value of the parameters, e.g., a small increase in K_2 will have a greater effect on reducing Z^* than a small increase in K_1 . Nonetheless, a user must again simultaneously consider the comparative costs involved in making finite changes, in conjunction with their expected effects, to arrive at an optimal marginal improvement based on this first order information. The sensitivity information is valuable, but requires some care in exploiting.

Table 4 gives the estimates of the first derivatives of the optimal reorder quantities Q_i and reorder points r_i with respect to each of the problem parameters. This is extremely detailed information which gives an indication of how the components of the solution vector itself will change as the various parameters change. In particular, this information can be used to obtain a first order estimate of the solution vector of a problem involving different parameter values, having obtained a solution for a given set of parameters.

The partial derivatives of the Lagrange multipliers with respect to the parameters are given in Table 5. Again, these can be used to obtain first order estimates of the Lagrange multipliers of a problem with different parameter values. In particular, the relative effects of the constraints on the optimal value of problems involving different parameter values can be estimated. Furthermore, it can be shown that the partial derivatives of the multipliers with respect to K_1 and K_2 yield the second partial derivatives of the optimal value function with respect to the parameters K_1 and K_2 , under the present conditions. Thus, the kind of information given in Table 5 can be used to provide a second order estimate of the optimal value function Z^* for different values of these parameters.

To illustrate and test the application of the type of information provided here, the first partial derivatives with respect to c_1 of the optimal value function Z^* , the solution components Q_i and r_i , and the Lagrange multipliers u_1 and u_2 , were used to give a first order (Taylor's Series) estimate of the corresponding solution values associated with the problem where the given value of c_1 was increased by one dollar. These estimates were compared with the respective values of the solution obtained by actually solving the perturbed problem. The results are summarized in Table 6. Though the perturbation is large (the parameter being increased by 100% of its current value), the estimates are seen to be extremely accurate with the exception of the estimated reorder quantity Q_1 . Many uses could be made of the estimated solution; e.g., should it be desirable to solve the perturbed problem accurately, it would be computationally extremely advantageous to use the estimated solution as a starting point.

The complete exploitation of this sensitivity analysis information now available will depend largely on user interest and ingenuity.

Table 4

SOLUTION POINT SENSITIVITY

PARTIALS	ITEM i		
	1	2	3
$\partial Q_i/\partial K_2$	- 47.3187	- 18.7610	- 14.9065
$\partial r_i/\partial K_2$	5.2265	6.1961	9.9671
$\partial Q_i/\partial c_1$	-208.8688	15.3140	14.3755
$\partial r_i/\partial c_1$	- 31.7918	- 10.3425	- 20.0020
$\partial Q_i/\partial \sigma_2$	- .8469	.2271	- .1084
$\partial r_i/\partial \sigma_2$	- .1273	1.0337	- .4919
$\partial Q_i/\partial c_2$	8.1908	- 4.9719	3.8522
$\partial r_i/\partial \sigma_2$	- 2.6783	- 7.2676	- 7.1087
$\partial Q_i/\partial \sigma_3$	- 1.2374	- .4033	.5843
$\partial r_i/\partial \sigma_3$	- .2611	- .3839	.0446
$\partial Q_i/\partial c_3$	- .1670	.4442	- .4251
$\partial r_i/\partial c_3$	- 2.3072	- 3.1702	- 12.1523

Table 5

L.M. SENSITIVITY

OPTIMAL L.M. DERIVATIVES		
PARTIALS	CONSTRAINT i	
	1: INVESTMENT	2: WORKLOAD
$\partial u_i^*/\partial K_2$	-.0002	-.1382
$\partial u_i^*/\partial c_1$	.0006	.1635
$\partial u_i^*/\partial \sigma_2$	.0000	.0006
$\partial u_i^*/\partial c_2$	.0002	.0489
$\partial u_i^*/\partial \sigma_3$	.0000	.0020
$\partial u_i^*/\partial c_3$	.0003	.0323

Table 6

FIRST ORDER ESTIMATES FOR A UNIT INCREASE OF PARAMETER c_1

QUANTITY $F(\epsilon^1)$	ESTIMATE $F(\epsilon^1)_1^+$	ACTUAL	% ABS. ERROR
$Z^*(\epsilon^1)$	15.159	14.996	1.08
$Q_1(\epsilon^1)$	324	412	21.36
$r_1(\epsilon^1)$	221	229	3.49
$Q_2(\epsilon^1)$	261	257	1.56
$r_2(\epsilon^1)$	267	268	.37
$Q_3(\epsilon^1)$	299	297	.67
$r_3(\epsilon^1)$	417	420	.71
$u_1(\epsilon^1)$	.0058	.0057	1.75
$u_2(\epsilon^1)$	.7865	.7671	1.94

$$^+F(\epsilon^1)_1 = F(\epsilon^0) + (\epsilon^1-\epsilon^0)^T\nabla_\epsilon F(\epsilon^0)$$
$$= F(\epsilon^0) + (1)\,\partial F(\epsilon^0)/\partial c_1\ .$$

REFERENCES

Armacost, Robert L. (1976). Computational experience with optimal value function and Lagrange multiplier sensitivity in NLP. Technical Paper Serial T-335. Program in Logistics, The George Washington University, Washington, D.C.

________, and Fiacco, Anthony V. (197[illegible]). Computational experience in sensitivity analysis for nonlinear programming. Mathematical Programming. 6:301-326.

________, and ______ (1975). Second-order parametric sensitivity analysis in NLP and estimates by penalty function methods. Technical Paper Serial T-324. Institute for Management Science and Engineering, The George Washington University, Washington, D.C.

________, and ______ (1976). NLP sensitivity for R.H.S. perturbations: A brief survey and recent second-order extensions. Technical Paper Serial T-334. Institute for Management Science and Engineering, The George Washington University, Washington, D.C.

________, and Mylander, W. Charles (1973). A guide to a SUMT-Version 4 computer subroutine for implementing sensitivity analysis in nonlinear programming. Technical Paper Serial T-287. Program in Logistics, The George Washington University, Washington, D.C.

Fiacco, Anthony V. (1973). Sensitivity analysis for nonlinear programming using penalty methods. Technical Paper Serial T-275. Institute for Management Science and Engineering, The George Washington University, Washington, D.C.

______, and McCormick, Garth P. (1968). Nonlinear Programming: Sequential Unconstrained Minimization Techniques. New York: Wiley.

McCormick, Garth P. (1972). Computational aspects of nonlinear programming solutions to large scale inventory problems. Technical Memorandum Serial TM-63488. Program in Logistics, The George Washington University, Washington, D.C.

Mylander, W. Charles, Holmes, Raymond L. and McCormick, Garth P. (1971). A guide to SUMT-Version 4: The computer program implementing the sequential unconstrained minimization technique for nonlinear programming. RAC-P-63, Research Analysis Corporation, McLean, Virginia.

Robinson, S. M. (1974). Perturbed Kuhn-Tucker points and rates of convergence for a class of nonlinear programming algorithms. Mathematical Programming. 7:1-16.

Schrady, D. A. and Choe, U. C. (1971). Models for multi-item continuous review inventory policies subject to constraints. Naval Research Logistics Quarterly. 18:541-563.

The Generalized Inverse in Nonlinear Programming -- Equivalence of the Kuhn-Tucker, Rosen and Generalized Simplex Algorithm Necessary Conditions*

L. Duane Pyle
University of Houston

1. Introduction

The equivalence of the Kuhn-Tucker and the Rosen Gradient Projection conditions for optimality was established by Mangasarian [4], who considered the following nonlinear programming problem formulation:

$$\text{Maximize } f(x)$$

$$\text{where} \quad H(x) = \begin{bmatrix} h_1(x) \\ \cdot \\ \cdot \\ \cdot \\ h_m(x) \end{bmatrix} \geq \theta, \; x \in E^n \quad , \tag{1.1}$$

and $h_1(x), \ldots, h_m(x)$ and $f(x)$ are concave, differentiable.

(θ is a vector of zeros.)

Except for minor changes in notation, the following, alternative, nonlinear programming problem formulation is discussed by Luenberger [3]:

$$\text{Minimize } f(x)$$

$$\text{where} \quad H(x) = \begin{bmatrix} h_1(x) \\ \cdot \\ \cdot \\ \cdot \\ h_s(x) \end{bmatrix} = \theta \quad ,$$

$$G(x) = \begin{bmatrix} g_{s+1}(x) \\ \cdot \\ \cdot \\ \cdot \\ g_{s+t}(x) \end{bmatrix} \geq \theta, \; x \in E^n \quad , \tag{1.2}$$

and $h_1(x), \ldots, h_s(x), g_{s+1}(x), \ldots, g_{s+t}$, and $f(x)$ are differentiable.

If $\hat{x}$ is a point satisfying the constraints

$$H(x) = \theta, \quad G(x) \geq \theta \tag{1.3}$$

$$\text{where} \quad g_j(\hat{x}) = 0 \quad \text{for} \quad j \in J_1 \quad ,$$

$$g_j(\hat{x}) > 0 \quad \text{for} \quad j \in J_2 \quad , \tag{1.4}$$

$$\text{and } J_1 \cup J_2 = \{s+1, \ldots, s+t\} \quad ,$$

then the constraints $h_1(x), \ldots, h_s(x)$, and $g_j(x)$ for $j \in J_1$, are said to be active at $\hat{x}$; the constraints $g_j(x)$ for $j \in J_2$ are said to be inactive at $\hat{x}$. A point, $\hat{x}$, satisfying (1.3) is said to be a regular point for the constraints (1.3) if the gradients of the constraints active at $\hat{x}$ are linearly independent.

The Kuhn-Tucker Conditions [2] are satisfied at the point $\hat{x}$ for the nonlinear programming problem (1.2) if there exist vectors $w^{(1)}$ and $w^{(2)}$ such that

$$\nabla f(\hat{x}) + [\nabla h_1(\hat{x}) \; \ldots \; \nabla h_s(\hat{x})] \, w^{(1)} + [\nabla g_{s+1}(\hat{x}) \; \ldots \; \nabla g_{s+t}(\hat{x})] \, w^{(2)} = \theta \tag{1.5}$$

where $(w^{(2)}, G(\hat{x})) = 0$, $w^{(2)} \leq \theta$ and

$$\nabla v(\hat{x}) = \begin{bmatrix} \frac{\partial v}{\partial x_1}(\hat{x}) \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial v}{\partial x_n}(\hat{x}) \end{bmatrix} \quad .$$

Luenberger [3] proves the following (necessary condition):

Theorem 1.1: Let $\hat{x}$ be a relative minimum point for the problem (1.2) and suppose $\hat{x}$ is a regular point for the constraints (1.3), then the Kuhn-Tucker conditions (1.5) are satisfied at $\hat{x}$.

Let

(1.6) $\mathcal{A}^T(\hat{x}) = [\nabla h_1(\hat{x}), \dots \nabla h_s(\hat{x}), \nabla g_{j_1} \dots, \nabla g_{j_p}(\hat{x})]$ be the n by (s+p) matrix whose columns consist of the gradients to the constraints which are active at $\hat{x}$; that is, $J_1 = \{j_1, j_2, \dots, j_p\}$. The Rosen Gradient Projection Conditions [10], [11] are satisfied at the point $\hat{x}$ for the nonlinear programming problem (1.2) if

$$[I - \mathcal{A}^+(\hat{x})\, \mathcal{A}(\hat{x})]\, \nabla f(\hat{x}) = \theta$$

(1.7) and

$$[(\mathcal{A}^+(\hat{x}))^T\, \nabla f(\hat{x})]_j \geq 0 \text{ for all } j \in J_1,$$

where $\mathcal{A}^+$ is the generalized inverse of $\mathcal{A}$ [4].

The expressions analogous to (1.7) formulated by Rosen, Mangasarian, and Luenberger may be obtained by making use of the relation

(1.8) $\mathcal{A}^+(\hat{x})^T = [\mathcal{A}(\hat{x})\, \mathcal{A}^T(\hat{x})]^{-1}\, \mathcal{A}(\hat{x})$

as, for example, in

(1.9) $I - \mathcal{A}^+(\hat{x})\mathcal{A}(\hat{x}) = I - \mathcal{A}^T(\hat{x})\,(\mathcal{A}^+(\hat{x}))^T$
$= I - \mathcal{A}^T(\hat{x})\,[\mathcal{A}(\hat{x})\,\mathcal{A}^T(\hat{x})]^{-1}\,\mathcal{A}(\hat{x}).$

Relation (1.8) provided the basis for the computational approach originally proposed by Rosen [10], who gave a procedure for "updating" $[\mathcal{A}(\hat{x})\,\mathcal{A}^T(\hat{x})]^{-1}$. Alternatives proposed by Gill and Murray [1] and Pyle [7], [8], [9], employ an orthonormal basis for the null space of $\mathcal{A}$ in obtaining a representation for $I - \mathcal{A}^+\mathcal{A}$. Motivation for the developments given in [9] was provided by the results of numerical experiments, involving randomly generated linear programming problems, interpreted in the context of the geometry of the simplex algorithm. This interpretation has led to the extension and refinement of the results given in [7] and [8] as reported in [9].

In this paper two results are presented. The first, provided for completeness, is an application of Mangasarian's approach, demonstrating the equivalence of the Kuhn-Tucker and the Rosen Gradient Projection Conditions as formulated for the problem (1.2). The second result establishes the equivalence of the Rosen Gradient Projection Conditions (1.7) and certain conditions which result from a natural extension of a generalization of the simplex algorithm [9] to the nonlinear programming problem.

2. Equivalence of the Kuhn-Tucker Conditions and the Rosen Gradient Projection Conditions

Suppose (1.7) holds at some point $\hat{x}$ which is regular for the constraints (1.3). Then

(2.01) $[I - \mathcal{A}^+(\hat{x})\, \mathcal{A}(\hat{x})]\, \nabla f(\hat{x}) =$
$[I - \mathcal{A}^T(\hat{x})\,(\mathcal{A}^+(\hat{x}))^T]\, \nabla f(\hat{x}) = \theta$

implies

(2.02) $\nabla f(\hat{x}) + \mathcal{A}^T(\hat{x})\, [-(\mathcal{A}^+(\hat{x})^T \nabla f(\hat{x})] = \theta$

Define $w^{(1)}$, $\bar{w}^{(2)}$ as follows:

(2.03) $\begin{bmatrix} w^{(1)} \\ \bar{w}^{(2)} \end{bmatrix} = -(\mathcal{A}^+(\hat{x}))^T\, \nabla f(\hat{x}),$

where $w^{(1)}$ has s elements and $\bar{w}^{(2)}$ has p elements. Then from

(2.04) $[(\mathcal{A}^+(\hat{x}))^T\, \nabla f(\hat{x})]_j \geq 0$ for all $j \in J_1$

it follows that $\bar{w}^{(2)} \leq \theta$. Now define

(2.05) $w^{(2)} = \begin{bmatrix} \bar{w}^{(2)} \\ \theta \end{bmatrix}$

where $w^{(2)}$ has t elements. (Note: G(x) in (1.2) is composed of t functions) $\bar{w}^{(2)} \leq \theta$ implies $w^{(2)} \leq \theta$, and from (2.02) and (2.03) and the form of $w^{(2)}$ it follows that

(2.06) $\nabla f(\hat{x}) + \mathcal{A}^T(\hat{x}) \begin{bmatrix} w^{(1)} \\ \bar{w}^{(2)} \end{bmatrix} =$

$$\nabla f(\hat{x}) + [\mathcal{A}^T(\hat{x}), \tilde{\mathcal{A}}^T(\hat{x})] \begin{bmatrix} w^{(1)} \\ w^{(2)} \end{bmatrix} = \theta,$$

where $\tilde{\mathcal{A}}^T(\hat{x})$ is a matrix of the gradients $\nabla g_j(\hat{x})$ for $j \in J_2$, corresponding to inactive constraints at $\hat{x}$. (2.06), together with

(2.07) $(w^{(2)}, G(\hat{x})) = (\begin{bmatrix} w^{(2)} \\ \theta \end{bmatrix}, G(\hat{x})) = 0$

where $w^{(2)} \leq \theta$, is (1.5).

Now, suppose (1.5) holds at some point $\hat{x}$ which is regular for the constraints (1.3). For simplicity, we adopt the notation

(2.08) $j_1 = s + 1, \dots, j_p = s + p$
where $J_1 = \{j_1, \dots, j_p\}$.

Letting $\hat{w}^{(1)}$ and $\hat{w}^{(2)}$ denote the vectors which satisfy (1.5), the relations

(2.09) $\hat{w}^{(2)} \leq \theta$, $G(\hat{x}) \geq \theta$ and
$(\hat{w}^{(2)}, G(\hat{x})) = 0$,

taken together, imply that

(2.10) $\hat{w}^{(2)}_{s+p+1} = \cdots = \hat{w}^{(2)}_{s+t} = 0,$

thus

$$\hat{w}^{(2)} = \begin{bmatrix} \bar{w}^{(2)} \\ \theta \end{bmatrix}$$

where $\bar{w}^{(2)} \leq \theta$ and $\bar{w}^2$ has p elements.
From (1.5)

(2.11) $\nabla f(\hat{x}) + [\nabla h_1(\hat{x}) \ \ldots \ \nabla h_s(\hat{x})] \ \hat{w}_1$

$$+ [\nabla g_{s+1}(\hat{x}) \ldots \nabla g_{s+p}(\hat{x})]\bar{w}^{(2)} = 0$$

or

(2.12) $\nabla f(\hat{x}) + \mathcal{A}^T(\hat{x}) \begin{bmatrix} \hat{w}^{(1)} \\ \bar{w}^{(2)} \end{bmatrix} = \theta .$

Since $\hat{x}$ is a regular point for the constraints (1.3), $\mathcal{A}^T(\hat{x})$ has full column rank, thus

(2.13) $(\mathcal{A}^T(\hat{x}))^+ = [\mathcal{A}(\hat{x})\mathcal{A}^T(\hat{x})]^{-1}\mathcal{A}(\hat{x}) = (\mathcal{A}^+(x))^T$

therefore,

(2.14) $(\mathcal{A}^+(\hat{x}))^T \ \nabla f(\hat{x}) + \begin{bmatrix} \hat{w}^{(1)} \\ \bar{w}^{(2)} \end{bmatrix} = \theta$

or, since $\bar{w}^{(2)} \leq \theta$,

(2.15) $[(\mathcal{A}^+(\hat{x}))^T \ \nabla f(\hat{x})]_j \geq 0$ for all $j \epsilon J_1$.

Solving (2.14) for $\begin{bmatrix} \hat{w}^{(1)} \\ \bar{w}^{(2)} \end{bmatrix}$ and substituting in (2.12), obtain

(2.16)
$$\nabla f(\hat{x}) - \mathcal{A}^T(\hat{x})(\mathcal{A}^+(\hat{x}))^T \ \nabla f(\hat{x})$$
$$= [I - \mathcal{A}^+(\hat{x})\mathcal{A}(\hat{x})] \ \nabla f(\hat{x}) = \theta.$$

(2.15) and (2.16) are the Rosen Gradient Projection Conditions (1.7), formulated for the nonlinear programming problem (1.2).

3. Equivalence of the Rosen Gradient Projection Conditions and the Generalized Simplex Algorithm Conditions

Consider the following nonlinear programming problem formulation:

Minimize $f(\hat{x})$

where

$$H(x) = \begin{bmatrix} h_1(x) \\ \cdot \\ \cdot \\ \cdot \\ h_s(x) \end{bmatrix} = \theta ,$$

(3.01)

$$G(x) = \begin{bmatrix} V(x) \\ x \end{bmatrix} \geq \theta,$$

$$V(x) = \begin{bmatrix} v_{s+1}(x) \\ \cdot \\ \cdot \\ \cdot \\ v_{s+r}(x) \end{bmatrix}, \ x \epsilon E^n ,$$

and $h_1(x), \ \ldots \ , h_s(x), v_{s+1}(x), \ \ldots \ ,$ $v_{s+r}(x)$ and $f(x)$ are differentiable.
(Note that any problem of the form (1.2) may be reformulated in the form (3.01).)

It is understood that $V(x)$, if not empty, consists of all the "coefficiented" inequality constraints; that is, $v_i(x) \neq x_j$ for $(j = 1, 2, \ \ldots \ , n)$.

Suppose $\hat{x}$ is a regular point for (3.01) and, for notational simplicity, that the active constraints, $V_1(x)$ from the set $V(x) = \begin{bmatrix} V_1(x) \\ V_2(x) \end{bmatrix} \geq \theta$ consist of the constraints $v_i(x)$ for $(i = s + 1, \ldots, m)$, and that the active constraints from the set $I x = x \geq \theta$ are the constraints $(u^{(j)}, x) = 0$ for $(j = m+k+1, \ \ldots \ , n)$, where $(k = 0, 1, \ \ldots \ , (n - m))$, and I is the n by n identity matrix composed of unit vectors $\{u^{(1)}, u^{(2)}, \ \ldots \ , u^{(n)}\}$. (Note: The special case $k = n - m$, $j = n+1, n$ corresponds to the situation where $\hat{x}_j > 0$ for $j = 1, 2, \ \ldots \ , n$.)

Define

(3.02) $A^T(\hat{x}) = [\nabla h_1(\hat{x}) \ldots \nabla h_s(\hat{x}) \nabla v_{s+1}(\hat{x}) \ldots \nabla v_m(\hat{x})]$, an n by m matrix whose columns consist of the gradients to the "coefficiented" constraints which are active at $\hat{x}_j$,

(3.03) $c(\hat{x}) = \nabla f(\hat{x}),$

(3.04) $b(\hat{x}) \ = \ [A(\hat{x})]\hat{x},$

and then denote the columns of $A(\hat{x})$ as follows:

(3.05) $A(\hat{x}) = [P_1(\hat{x}) \ \ldots \ P_n(\hat{x})].$

Consider the following linear programming problem approximation of (3.01) at $\hat{x}$:

Minimize $(x, c(x))$

(3.06) where

$$[A(\hat{x})]x = b(\hat{x})$$
$$x \geq \theta.$$

The developments given in [9] for the linear programming problem may be applied to (3.06):

In summary, letting

(3.07) $B(\hat{x}) = [P_1(\hat{x}) \ \ldots \ P_m(\hat{x})]$, $\hat{x}$ a regular point for (3.01) implies $B(\hat{x})$ is nonsingular. Forming

$$(3.08)\quad [B(\hat{x})]^{-1} A(\hat{x}) = [I\ \bar{P}_{m+1}(\hat{x}) \dots \bar{P}_n(\hat{x})]$$

obtain an orthonormal basis, $\{\eta^{(m+i)}(\hat{x})\}$ for $(i = 1, 2, \dots, k)$, for the subspace in E^n which is spanned by the linearly independent columns of the following matrix:

$$(3.09)\quad F(\hat{x}) = \begin{bmatrix} R(\hat{x}) \\ I \\ 0 \end{bmatrix} =$$

$$\begin{bmatrix} -\bar{P}_{m+1}(\hat{x}) & -\bar{P}_{m+2}(\hat{x}) & \dots & -\bar{P}_{m+k}(\hat{x}) \\ 1 & 0 & & 0 \\ 0 & 1 & & \vdots \\ \vdots & 0 & & 1 \\ & \vdots & & 0 \\ & & & \vdots \\ 0 & 0 & & 0 \end{bmatrix}$$

where $R(\hat{x}) = -[\bar{P}_{m+1}(\hat{x}) \dots \bar{P}_{m+k}(\hat{x})]$ is m by k, I is k by k and 0 is $[n-(m+k)]$ by k.

The Rosen Gradient Projection Conditions (1.7), formulated for problem (3.01) in accordance with the notational conventions assumed above, involve the matrices

$$(3.10)\quad \mathcal{A}(\hat{x}) = \begin{bmatrix} A(\hat{x}) \\ \mathcal{U} \end{bmatrix} \text{ and } I-\mathcal{A}^{+}(\hat{x})\mathcal{A}(\hat{x})$$

where $\mathcal{U}^T = [u^{(m+k+1)}, \dots, u^{(n)}]$.

Now, $\mathcal{A}(\hat{x})\, F(\hat{x}) = 0$ by construction and the k columns of $F(\hat{x})$ form a basis for the null space of $\mathcal{A}(\hat{x})$, thus

$$(3.11)\quad I-\mathcal{A}^{+}(x)\mathcal{A}(x) = \sum_{i+1}^{k} \eta^{(m+i)}(\hat{x})\, \eta^{(m+1)}(\hat{x})^T .$$

The developments given in [9] are thus seen to provide an alternative method for obtaining the orthogonal projection matrix

$[I - \mathcal{A}^{+}(\hat{x})\, \mathcal{A}(\hat{x})]$.

The equivalence of the Rosen Gradient Projection Condition, $[(\mathcal{A}^{+}(\hat{x}))^T \nabla f(\hat{x})]_j \geq 0$ and the analogous construction obtained using the generalization of the simplex algorithm given in [9], **follows from** theorem 3.1.

Theorem 3.1: Let

$$(3.12)\quad \overset{k}{\bar{c}}_j = \left(\mathcal{A}^{+}(\hat{x})\mathcal{A}(\hat{x})c(\hat{x}), \begin{bmatrix} B^{-1}(\hat{x})u^{(j)} \\ \theta \end{bmatrix}\right)$$

where $j = 1, 2, \dots, m$

$k = 1, 2, \dots, (n - m + 1)$

and

$$(3.13)\quad \overset{k}{\bar{c}}_j = \left(\mathcal{A}^{+}(\hat{x})\mathcal{A}(\hat{x})c(\hat{x}), \begin{bmatrix} -\bar{P}_j \\ \theta_{1_j} \\ 1 \\ \theta_{2_j} \end{bmatrix}\right)$$

where $j = m + k, \dots, n$

$k = 1, 2, \dots, (n - m)$

and the "j"th element in the vector

$$\begin{bmatrix} -\bar{P}_j \\ \theta_{1_j} \\ 1_j \\ \theta_{2_j} \end{bmatrix}$$

is unity.

Then

$$(3.14)\quad \overset{k}{\bar{c}}_j = [(\mathcal{A}^{+}(\hat{x}))^T c(\hat{x})]_j$$

for $j = 1, 2, \dots, m$

$k = 1, 2, \dots, (n - m + 1)$

and

$$(3.15)\quad \overset{k}{\bar{c}}_j = [(\mathcal{A}^{+}(\hat{x}))^T c(\hat{x})]_{j-k+1}$$

for $j = m + k, \dots, n$

$k = 1, 2, \dots, (n - m)$.

Proof: Upon substituting $\mathcal{A}^T(\hat{x})(\mathcal{A}^{+}(\hat{x}))^T$ for $\mathcal{A}^{+}(\hat{x})\mathcal{A}(\hat{x})$ in (3.12) and (3.13), respectively, obtain

$$(3.16)\quad \overset{k}{\bar{c}}_j = \left(\mathcal{A}^T(\hat{x})(\mathcal{A}^{+}(\hat{x}))^T c(\hat{x}), \begin{bmatrix} B^{-1}(\hat{x})u^{(j)} \\ \theta \end{bmatrix}\right)$$

$$= \left((\mathcal{A}^{+}(\hat{x}))^T c(x), \mathcal{A}(\hat{x}) \begin{bmatrix} B^{-1}(\hat{x})u^{(j)} \\ \theta \end{bmatrix}\right)$$

$$= \left((\mathcal{A}^{+}(\hat{x}))^T c(\hat{x}), \begin{bmatrix} u^{(j)} \end{bmatrix}\right) = [(\mathcal{A}^{+}(\hat{x}))^T c(\hat{x})]_j$$

where $j = 1, 2, \dots, m$

$k = 1, 2, \dots, (n - m - 1)$,

which is (3.14); and

$$(3.17)\quad \overset{k}{\bar{c}}_j = \left(\mathcal{A}^T(\hat{x})(\mathcal{A}^{+}(\hat{x}))^T c(\hat{x}), \begin{bmatrix} -\bar{P}_j \\ \theta_{1_j} \\ 1 \\ \theta_{2_j} \end{bmatrix}\right)$$

$$= \left((\mathcal{A}^{+}(\hat{x}))^T c(\hat{x}), \mathcal{A}(\hat{x}) \begin{bmatrix} -\bar{P}_j \\ \theta_{1_j} \\ 1 \\ \theta_{2_j} \end{bmatrix}\right)$$

$$= ((\mathcal{A}^{+}(\hat{x}))^T c(\hat{x}), u^{(j)})$$

$$= [(\mathcal{A}^{+}(\hat{x}))^T c(\hat{x})]_{j-k+1}$$

where $j = m + k, \dots, n$

$k = 1, 2, \dots, (n - m)$,

which is (3.15).

Remarks: (i) The vectors $\begin{bmatrix} B^{-1}(\hat{x})u^{(j)} \\ \theta \end{bmatrix}$

and

$$\begin{bmatrix} -\bar{P}_j(\hat{x}) \\ \theta_{1_j} \\ 1 \\ \theta_{2_j} \end{bmatrix} = \begin{bmatrix} -B^{-1}(\hat{x})\ P_j(\hat{x}) \\ \theta_{1_j} \\ 1 \\ \theta_{2_j} \end{bmatrix}$$

both involve $B^{-1}(\hat{x})$, and this permits computation of the $\bar{c}^k_j$ values by a Revised Simplex Algorithm approach, as described in [9]. Also, see [5] where the matrix $F(\hat{x})$ is used directly in implementing the method of reduced gradients.

(ii) In [9] the vectors $\{\eta^{(m+i)}\}$ for $(i=1, \ldots, k)$ were obtained sequentially by an application of the Gram-Schmidt orthogonalization procedure. From (3.12) and (3.13), it follows that the individual vectors, $\eta^{(m+i)}$, are not required in realizing the Rosen Conditions, thus any other method (e.g. [1]) for generating the orthogonal projection having a range equal to the column space of $F(\hat{x})$ could be utilized.

(iii) Note that in (3.12) and (3.16), $u^{(j)}$ denotes an m by 1 unit vector, whereas in (3.17) $u^{(j)}$ denotes an n by 1 unit vector.

4. Bibliography

[1] Gill, P.E., and W. Murray, eds. *Numerical Methods for Constrained Optimization*, Academic Press (1974).

[2] Kuhn, H.W., and Tucker, A.W., *Nonlinear Programming*, Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, University of Calif. Press (1951).

[3] Luenberger, D.G., *Introduction to Linear and Nonlinear Programming*, Addison-Wesley (1973).

[4] Mangasarian, O.L., *Equivalence in Nonlinear Programming*, Naval Research Logistics Quarterly, Vol. 10, #4 (Dec. 1963).

[5] Murtagh, B.A., and M.A. Saunders, *Nonlinear Programming for Large, Sparse Systems*, Tech. Rpt. SOL 75-15, Systems Optimization Laboratory, Stanford University (August 1976).

[6] Penrose, R., *A Generalized Inverse for Matrices*, Proc. Cambridge Philos. Soc., Vol. 51 (1955).

[7] Pyle, L.D., *A Simplex Algorithm-Gradient Projection Method for Nonlinear Programming* (Abstract) ACM Proceedings National Conference, Chicago (1971).

[8] Pyle, L.D., *A Simplex Algorithm-Gradient Projection Method for Nonlinear Programming*, CSD TR 55, Purdue University, Dept. of Comp. Sc., (1971).

[9] Pyle, L.D., *The Generalized Inverse in Linear Programming - A Generalization of the Simplex Algorithm*, Proceedings IX International Symposium on Mathematical Programming, North Holland (to appear).

[10] Rosen, J.B., *The Gradient Projection Method in Nonlinear Programming, Part I, Linear Constraints*, J. SIAM, Vol. 8 (1960).

[11] Rosen, J.B., *The Gradient Projection Method in Nonlinear Programming, Part II, Nonlinear Constraints*, J. SIAM, Vol. 9 (1961).

***This research supported by National Science Foundation Grant DCR74-17282**

TEACHING MATHEMATICAL PROGRAMMING TO THE CONSUMER

Marshall L. Fisher
Decision Sciences Department
The Wharton School
University of Pennsylvania

Most courses on mathmatical programming are taught by producers (i.e., researchers who are developing mathematical programming technology) to potential consumers of that technology. The pedagogical approach that is most natural for a producer may not be at all natural for a potential consumer, a fact that explains, in part, why many potential consumers fail to become actual consumers. This paper is concerned with the particular needs of the consumer.

I will provide a number of specific suggestions for course content and design. These suggestions are based primarily on my experiences teaching a fundamental course on mathematical methods for decision making offered by the Decision Sciences Department of the Wharton School. This course, which devotes 10 weeks to linear and integer programming, is taken by about 200 of the 600 students that enter the M.B.A. program each year. The backgrounds of the students vary widely. This year my two sections had a total of 70 students that included 12 math majors, 30 engineering and science majors, 18 management and economics majors and 10 majors in various non-technical areas like English, Anthoropology, Philosophy and Language. Most of these students do not currently plan to become specialists in Mathematical Programming. Many have not yet decided on a major, but of those who have 31 plan to specialize in Finance.

Those who preach mathematical programming should also practice its tenets. Certainly the viewpoint of constrained optimization provides a useful conceptual framework for the problem addressed by this paper--designing a course on mathematical programming for a specified purpose. I will use this framework informally in developing my suggestions. I will take as an objective that the course should provide maximum benefit to the future careers of the students. Benefits can range from changing the way the students think about their work to motivating and enabling them to formulate and solve mathematical programming models of problems forced in their careers. Constraints are imposed on the activities we can perform to optimize this objective by the limited teaching time available, by the students diverse and in some cases, limited mathematical backgrounds, and by their long term career plans that frequently do not place primary importance on technical expertise in mathematical programming.

I will now list seven recommentations for the design of a course that "solves" the "optimization problem" I have just outlined. All suggestions are admittedly subjective, but I believe they follow logically from the framework I have given.

1. Methodological topics should be selected to maximize the ratio of their short run benefits to their cost.

By 'short run benefits' I mean the ability to operationally solve real problems and by 'cost' I mean the intellectual prerequisites and time requirements to understand the topic. This criteria leads to the selection of topics like linear programming, seperable programming, heuristics and branch and bound methods for integer programming models, and the special algorithm for transportation problems. It requires the exclusion of many fascinating topics in nonlinear and integer programming that (a) would require too much time to teach or (b) have too great a mathematical prerequisite or (c) lie on the frontier of research and thus have not yet demonstrated their practical usefulness or (d) have all of the above drawbacks. I would list as borderline cases Lagrangian relaxation and an informal treatment of the analysis of heuristics and algorithms for combinatorial optimization problems.

2. Algorithms should be taught in a way that conveys their logic but avoids technicalities.

One good method for doing this is to simply demonstrate the algorithm on a small example derived from a real application. The students usually have an intuition about the example that can be used to explain the logic of the algorithm. This excercise can be followed with a verbal statement of the algorithm for the general case.

A variation on this approach is to present the students with an example and ask them to use their common sense to develop a solution procedure. Frequently the students will be able to discover the essential features of an algorithm themselves and at a minimum this experience provides a good perspective for understanding a more formal presentation of the algorithm. For example, when confronted with a problem of shipping amounts of a product from several supply points to several demand points most students will quickly devise a tableau format for recording possible solutions and discover a feasible solution using some varient of the Northwest corner rule. With a little coaching they will make improving adjustments in this solution until an optimal solution is obtained. It is then only necessary to show them that their adjustment procedure can be systemized and to provide an optimality test and proof.

3. Use articles that report application success stories to show how mathematical programming can be used.

The increased emphasis on the publication of solid application work has produced a wealth of articles that are highly suitable for classroom discussion. Journals like *Interfaces* are ideal sources for articles of this type. Recent examples of applications articles in this journal include a discussion of inventory planning for a mining company (Reddy[6]) and a description of how to win an election with linear programming (Southwick and Zionts[7])-an article that is sure to be a hit in any year divisible by 4. *Management Science*, *Operations Research* and the *Harvard Business Review* are also not to be overlooked for articles suitable for this purpose. The paper by Klingman, Randolph and Fuller[4] is highly recommended for a real-life illustration of branch and bound methods combined with transportation or network flow models for a location analysis problem. Geoffrion[2] gives an excellant management-oriented discussion of integer programming methods for warehouse location.

4. Use cases to develop modeling skills.

The ability to construct a model of a complicated process is a skill best acquired through practice. Case study problems are an excellent way for providing students with this practice if the cases are appropriately selected. The case should present a complicated, messy, and apparently baffling decision problem. At the same time, the problem should have an underlying structure and sufficient data should be included to allow the student to discover and model the structure and eventually arrive at a satisfying resolution of the original problem. The feeling of having 'created order out of apparent chaos that a student experiences from analyzing such a case is the single thing most likely to 'turn him on' to the use of mathematical programming.

Errors of two types should be avoided in the selection of cases. Many cases are too structured, really just expanded formulation exercises. Others that have been designed for policy discussion are too unstructured and don't provide enough data for the student to sink his teeth into. Good sources for cases that strike a balance between these extremes are books like Berry and Whybark[1] and Von Lanzenauer[8]. I have also had considerable success with cases developed from my own or another faculty member's consulting experiences.

Students in the Wharton course described earlier analyze cases in teams. Each team is selected to have some students with technical backgrounds and some with non-technical backgrounds. The results of each team analyses are presented in a written report and an oral report to the class. The Wharton Communications Consultants, a service group at Wharton, provide expert coaching and video taping equipment to assist the teams in their oral presentations.

5. Provide computer codes so that case analyses may be pursued to completion.

The computer codes should be easy to use yet still solve reasonably large problems. Ideally they should be available on an interactive computer. We have had excellent success with the package of programs described in Land and Powell[5]. The package has an simple input and output, provides linear, integer, and parameter programming capabilities and is provided to the students interactively on a DEC-10 computer.

6. Bring in Outside Speakers to provide a feel for the 'real world.'

Staff OR groups and consulting organizations are frequently involved in interesting applications of mathematical programming and they are quite happy to talk about them. Frequently these organizations are also involved in recruiting efforts on campus and view speaking to classes as a desireable supplement to these activities. They can provide breadth, background information, and a feel for the real world that is unavailable from other sources. Examples like that of one company which routinely solves a 50,000 variable 10,000 constraint LP model of its corporate activities have a dramatic impact and provide additional evidence that mathematical programming is not only useful but used. Ideally, the particular speakers chosen should be related to the applications articles and cases used.

7. Emphasize the intuitions about complex systems that mathematical programming can reveal.

Complex systems are usually difficult to understand and an unsystematic approach frequently leads to intuitions that are incorrect. Mathematical programming models and algorithms can enhance ones intuition about a complex system if the model solution is dissected 'a posteriori' to determine why a given answer was obtained (see Geoffrion[3] for a discussion and examples on this point).

There are at least two reasons why a manager should develop good intuitions about a system he manages. Firstly, if he has the intuition to understand why a particular model solution was obtained, then he will have more faith in the solution. Secondly, the actions prescribed by the model solution may be only a fraction of the required decisions that relate to the system being analyzed. Frequently, managers must also make many related operating decisions that were too numerous and detailed to include explicitly in the model. Moreover, even for decisions which are included in the model, the results must often be manually adjusted to allow for intangible aspects of the system that were not included. If managers have used mathematical programming methods to obtain a better intuitive understanding of their system, then they will do a better job of making these additional decisions.

Students will appreciate this phenomenon if they are lead to discover the reasons for answers obtained in cases and applications articles. Appropriate examples are numerous. For the transportation problem, the optimal shadow prices can reveal the initially surprising fact that shipping more product will in some cases reduce shipping costs. The linear programming model described in Reddy[6] lead to the discovery that customer orders should be satisfied with relatively low grade product even though this generated less revenue per sale. This policy completely contradicted existing folklore within the company, but was found to be correct because of the scarcity of high grade product.

My experience has been that students are much more satisfied with the idea that mathematical programming methods can enhance their own intuition and reasoning power than they are with a viewpoint that portrays an algorithm like the simplex method as a giant wrench with which they can go around tightening loose corporate nuts.

REFERENCES

1. W.L. Berry and D.C. Whybark, Computer Augmented Cases in Operations and Logistics Management, South-Western Publishing Co., 1972.

2. A.M. Geoffrion, "Better Distribution Planning with Computer Models," Harvard Business Review, (July-August, 1976).

3. A.M. Geoffrion, "The Purpose of Mathematical Programming is Insight, Not Numbers," Working Paper No. 249, Western Management Science Institute, UCLA, June, 1976.

4. D. Klingman, P.H. Randolph, S.W. Fuller, "A Cotton inning Problem," Operations Research, ol.24, No.4, (July August, 1976).

5. A. Land and S. Powell, Fortran Codes for Mathematical Programming, John Wiley and Sons, 1973.

6. J.M. Reddy, "A Model to Schedule Sales Optimally Blended from Scarce Resources," *Interfaces*, Vol.6, No.1, Pt.2, pp 97-107, (November 1975).

7. L. Southwick, Jr., and S. Zionts, "Optimal Resource Allocation in a Local Election Campaign," *Interfaces*, Vol.6, No.1, pp 53-63, (November 1975).

8. C.H. von Lanzenauer, *Cases in Operations Research*, Research and Publication Division, School of Business Administration, The niversity of Western Ontario, London, Canada, 1975.

ON TEACHING LINEAR PROGRAMMING FUNDAMENTALS

John M. Mulvey
Roy D. Shapiro
Harvard University
Graduate School of Business Administration

Abstract

This paper provides an overview of the methods employed by the Harvard Business School in teaching linear programing. Specifically, we deal with the pedagogical aspects of teaching linear programming fundamentals to predominatly nonmathematical MBA students. These students will eventually become decision makers in government and business, and since many of their decisions will require the use of complex mathematical models, it is crucial that these individuals thoroughly understand the fundamentals of the application of quantitative tools. The cornerstones of our pedagogy are presented: 1) the use of a flexible interactive linear programming system, 2) the active participation of faculty and students in class discussion, 3) the simulation of semi-realistic situations (cases) in which linear programming has been employed, and 4) a carefully prepared set of course materials. With these elements, we believe that learning is enhanced and that the students are well prepared to successfully use linear programming techniques. A detailed description of the program is provided.

Although quantitative techniques have become an integral component of most MBA curricula, there have been few published papers which deal with the pedagogical aspects of quantitative courses. To be sure, there have been numerous articles on general management science education, for example, the special issue of Management Science on Educational Issues [2], Wagner's talk on the ABC's of OR [6], and several papers [1, 3, 5] which have described the content of various MS/OR and related curricula. By and large, however, these do not address the specific issues of pedagogy.

Pedagogy is particularly relevant today for several reasons. First of all, the field of management science, including linear programming, is entering a maturation phase in which fundamental concepts are being refined and polished. Applications are assuming an increasingly important role. In addition, many of tomorrow's decision makers who will assume positions in which they are able to make use of analytical tools do not possess mathematical background or inclination. Thus, we might lose important opportunities for applying management science if we do not, or cannot, teach non-quantitative decision makers how to properly utilize these analytical tools. See Healy [4]. If management science is to continue growing, we must develop reliable methods for communicating its ideas and tenets.

The lack of a suitable pedagogical framework is most noticable when discussing the teaching of analytical tools to students who do not possess mathematical background. This article attempts to fill this void by addressing the specifics of teaching linear programming fundamentals to non-quantitative MBA students. Herein, we use the pedagogical device of interactive exposition; a technique which is repeatedly employed by the authors in the classroom. Although this report does not strictly follow the "case" format, it does describe the way in which linear programming is taught at the authors' institution, i.e., Harvard Business School (HBS).

The primary issue which is addressed in this article is: how can linear programming be taught so that it will be an effective planning tool which will be 1) understood and 2) used? The authors believe that an interactive teaching method promotes substantially more usable and longer lasting learning than the traditional lecture method, especially among students who will become tomorrow's general managers. Often, in a lecture-oriented classroom, the norm requires that the student be attentive and carefully transcribe the instructor's words from blackboard to notebook. In-class decision making is rarely required. This is reserved for take-home problem sets and (usually only after hours of "cramming") exams. Often the student forgets much of the content of the course shortly after the exam is over. In our experience, this happens less frequently if the student

- learns by analyzing actual or near-replicas of business situations in which linear programming has played a part,
- actively participates in class discussions of the analysis, implementation, and related managerial issues, and
- has easy access to an interactive, on-line computer package used in analyzing these situations.

Since the student is at all times personally involved in the learning process, his tendency is to integrate the concepts discussed and thus to retain them.

The remainder of this article will describe the linear programming segment of the required quantitative methods (Managerial Economics) course taught in the first year of the MBA program at Harvard. Section 1 will give a short overview of the school's objectives, student characteristics, and structure of the first-year program. Section 2 will detail the initial case used in this segment (the durable Sherman Motors case) and the pedagogy involved in teaching the day-long introduction to linear programming. Section 3 will describe the next four sessions in which the students are gradually presented with more involved and realistic situations. Since an important aspect of the learning experience is the interactive linear programming package that is employed, the capabilities of this package will be described. This report is largely descriptive in nature so that the reader can fully appreciate the unusual constraints which arise at HBS and how the ensuing difficulties are overcome.

1. Overview: Philosophy and Structure

The Harvard Business School MBA program gives men and women training for line management positions. The intent is to produce generalists, not specialists, nor technical staff personnel, so that neither mathematics nor theory is stressed. For example, the simplex method *per se* is not taught. (Graphically, the vertex-following concept is demonstrated, but the details of the algorithm or the theory are not discussed.) The emphasis is instead on formulation of appropriate models and the subsequent analysis of the economic and managerial implications of the solution results. The rationale for this is clear: our belief is that general managers have no need to know how to pivot, wheras, in order to effectively use linear programming, the interpretation of results is essential. (In fact, we suggest that the reader try to recall the last time at which he needed to perform a pivot by hand!)

In the first year of the MBA program, all students are enrolled in the same courses. The approximately 800 incoming students are divided into nine sections each of which will remain together in the same room for all their classes throughout the first year. By the sixth week, each section has started to develop the tight-knit social fabric which so characterizes the Harvard Business School first-year section. This, even more than the well-known case method, makes participatory education possible. The student, surrounded by peers whom, for the most part, he trusts and respects, is willing to go out on a limb, try new directions and experiences. The class size, approximately 85, insures that all points of view will be represented and that most important issues will emerge.

Students come from a wide range of geographical areas with an even broader variety of backgrounds and experiences. This diversity also helps to keep class discussion interesting. The majority of the class enter with at least two years of business experience. There is no mathematics prerequisite and, typically, previous training in mathematics ranges from Ph.D's (few) to none-since-tenth-grade (more frequent). One might believe that this situation might inevitably lead to the poorly-trained few being lost or to the highly-trained becoming bored, but the attributes of the participatory learning can prevent this. For example, in the first or second week of the year, one of the students with a higher degree of technical expertise will inevitably make a remark involving a complex or theoretical concept not understood by most. He might then be asked to role-play: one of the class's "poets" will be cast as his "boss" and ask him to explain his comment. Often he tries futilely. This drives home the lesson that the ability to explain one's technical methods to those with less mathematical training is as crucial in the business world as are those methods themselves. As the year progresses, the different learning roles become more clear: the "poets" concentrate on content - new concepts and techniques; the "engineers" refine and abstract and learn to communicate their knowledge.

The core quantitative methods course consists of five segments taught over a six-month period: decision analysis (including the basics of probability), forecasting (mainly multiple linear regression), simulation, competitive decision making (elementary game-theoretic concepts), and linear programming. The next two sections will describe this latter segment.

2. Managerial Economics Day

The introductory sessions on linear programming, euphemistically called Managerial Economics Day, take up an entire day for the first-year MBA's, during which interactive sessions are interspersed with discussion sessions. Our objectives are three-fold:

1) to introduce students to the recognition of situations in which linear programming might be useful,
2) to indicate the basic structure of a linear programming model, and
3) to introduce students to the interpretation of linear programming solutions through an interactive computer package.

The case used in this introductory day is Sherman Motors*, an example adapted from Robert Dorfman's "Mathematical or 'Linear' Programming: A Nonmathematical Approach," *American Economic Review*, December 1953. The example is a simple resource allocation problem involving a motor manufacturing firm with two truck models, model 101 and 102, to be produced using four kinds of capacity: metal stamping, engine assembly, 101 assembly and 102 assembly. Given the data in the case, the linear programming model below can be formulated.

$$\begin{array}{ll} & \text{maximize } \; 300\,N_1 + 350\,N_2 \\ \text{subject to} & \\ \text{metal stamping:} & N_1 + 5/7\,N_2 \le 2500 \\ \text{engine assembly:} & N_1 + 2\,N_2 \le 3333 \\ \text{101-assembly:} & N_1 \le 2250 \\ \text{102-assembly:} & N_2 \le 1500 \\ & N_1, N_2 \ge 0 \end{array}$$

where N_1 and N_2 are, respectively, the monthly production of 101's and 102's. This simple model is depicted graphically in the case and students, with no previous introduction, are assigned the task of deciding what to do with it.

The example provides a good transition from situations involving uncertanties but few complex constraints, to those where the uncertainty is minor, but the constraints are essential. The simple formulation permits the easy discussion of basic concepts such as decision variables, objective function, feasible set, and so on. Many students analyze the graphical formulation by suggesting some vertex comparing strategy (in this example, enumeration is easy enough) and this allows the introduction of the vertex-following concept.

At this point, after a general discussion of the issues of linear programming and a more specific discussion on the formulation of the Sherman Motors example, the students are introduced to the interactive computer package they will be using and sent to the terminals for their first set of exercises. A brief description of that program is needed here.

CLP*, conversational linear programming, as it is called, operates by asking a number of directed questions and allowing the user to sequentially request a series of options. It allows the user to input a linear programming model, to edit and file the specifications, and to receive output in several ways. The output for the Sherman Motors example is given in Exhibit I. Output section 1 gives the optimal solution; section 2 the values of the slack and surplus variables; section 3 the value of the dual variables or shadow prices; section 4 the reduced costs for non-basic variables, i.e., the decrease in the objective which would occur if the variable was introduced into the basis at a unit level. Output sections 5 and 6 give ranges on the objective function and right hand side values for purposes of sensitivity analysis. Section 5 gives ranges on the coefficients of the objective function such that the optimal solution remains unchanged as long as the coefficient remains within that range. Similarly, section 6 gives ranges on the right hand side values within which the shadow prices are constant.

The first set of computer exercises is given in Exhibit II. Their purpose is to introduce the students to the conversational linear programming package - input, edit, and optimize, and, more importantly, to present, in an *experiential* way, the concept of a shadow price. Thus, in exercises 3 and 4, the students are asked to change the amount of the available engine capacity from 3333 to 3334, then to 4333, then to 4433. The first change causes an increase in the objective function equal to the shadow price for that constraint, the second change causes an increase of 1000 times the shadow price, and the third change, which takes the right hand side value outside the range prescribed (see Exhibit I, output section 6) and therefore requires a basis change, causes an increase of less than the 1100 times the shadow price which the students expect. Most students when examining the output, which has not yet been interpreted for them, will

---

*Intercollegiate Clearing House #6-107-010

*This package was developed by Stephen Bradley. For more information, see "Using the Conversational Linear Programming System," ICH #9-172-240

Exhibit I

CLP OUTPUT FOR SHERMAN MOTORS MODEL

```
TITLE: SHERMAN MOTORS
PROCEED, DISPLAY, OR REJECT? D

OBJECTIVES:

             N101      N102

CONTRIB      300.0     350.0

CONSTRAINTS:

             N101      N102        RELATION  CAPACITY

STAMPING     1.000     .7143         LE       2500.
ENGINE       1.000     2.000         LE       3333.
ASMB101      1.000     .0000         LE       2250.
ASMB102      .0000     1.000         LE       1500.

PROCEED OR REJECT? P

MAXIMIZE OR MINIMIZE? MAX

OPTIMAL SOLUTION FOUND.
     CONTRIB      =837964.

OUTPUT OPTION? 0

ALL ITEMS NOT LISTED IN SECTIONS 1 - 6 HAVE THE VALUE
ZERO.
*1* DECISION VARIABLES
  1. N101        2037.51
  2. N102        647.745

*2* SLACK(+) AND SURPLUS(-) IN CONSTRAINTS
  3. +ASMB101   212.495
  4. +ASMB102   852.255

*3* SHADOW PRICES FOR CONSTRAINTS
  1. STAMPING   194.401
  2. ENGINE     105.599

*4* REDUCED COSTS FOR DECISION VARIABLES

*5* RANGES ON COEFFICIENTS OF OBJECTIVE CONTRIB
    VARIABLE  LOWER BOUND  CURRENT VALUE  UPPER BOUND
  1. N101       175.00        300.00        490.00
  2. N102       214.29        350.00        600.00

*6* RANGES ON VALUES OF RIGHT-HAND-SIDE CAPACITY
    CONSTRAINT  LOWER BOUND  CURRENT VALUE  UPPER BOUND
  1.  STAMPING    1404.5        2500.0        2636.8
  2.  ENGINE      2950.3        3333.0        4428.6
  3.  ASMB101     2037.5        2250.0        UNBOUNDED
  4.  ASMB102     647.74        1500.0        UNBOUNDED
```

Exhibit II

SHERMAN MOTOR COMPANY

Note: For each of these questions, start from the basic assumptions of the case. (Do not carry over additional assumptions from one numbered question to the next.)

1. Find the best product mix for Sherman Motors under the basic assumptions of the case.
2. Find the best product mix if it is found that the stamping department capacity if 3,500 Model 102's (as in the case) but only 2,000 Model 101's.
3. What is the best product mix if engine assembly capacity is raised to 3334? What is an extra unit of engine assumbly capacity worth?
4. What are 1,000 additional units of Model 101 equivalent engine assembly capacity worth? What about 1,100 units?

notice that the number listed under shadow price for engine assembly capacity is the same as the increase in the objective function which they have just observed.

After the completion of this first computer exercise, there is a return to the classroom for a 45-minute class session in which the instructor answers questions and reinforces the shadow price concept. The students then return to the terminals for a second set of exercises as shown in Exhibit III. It is more complex than the first and requires reformulation rather than simply revision of coefficients. The first question introduces a third product into the production process. It is important to note that this new truck is not produced in the revised optimal solution. The students are thus presented, again experientially rather than directly, to the notions of reduced cost and sensitivity on the objective function coefficients. The second question requires a complete reformulation with the addition of two new variables and a new constraint. It provides the framework for a concluding discussion of the CLP output when the students return for their third class session. Exhibit IV gives a typical schedule for this introductory linear programming day.

The student's reactions to this method are positive. Linear programming is a topic which the MBA's at first

Exhibit III

SECOND SET OF EXERCISES

Note: For each of these questions, start from the basic assumptions of the case. (Do not carry over additional assumptions from one numbered question to the next.)

1. Sherman Motors is considering introducing a new economy truck, to be called Model 103. The total metal stamping capacity would be sufficient for 3,000 Model 103's per month, while the total engine assembly shop would be enough for 2500 Model 103's. The 103's could be assembled in the 101 assembly department; each 103 would require only half as much time as a 101. Each Model 103 would give a contribution of $225.
   a) Formulate the production decision with the three trucks as a linear programming problem and then solve the problem with MBALP to verify that no Model 103's should be produced.
   b) How much would it cost in terms of contribution if, for some other reasons, management insisted that at least one Model 103 be made?
   c) How high would the contribution on each 103 have to be before it became attractive to produce the new model?
2. The engine assembly line can be put on overtime. Suppose that efficiencies don't change and that 2000 units of overtime capacity are available. If direct labor costs increase by 50% on overtime and if the fixed overhead on the line on overtime is $40,000, the variable overhead remaining the same, would it pay to go on overtime?

Exhibit IV

TYPICAL CLASS SCHEDULE FOR M.E. DAY

9:15-10:30	First Class Session
10:45-11:45	First Computer Session
11:45-12:30	Second Class Session
12:30- 2:00	Lunch and Preparation of Second Exercise Set
2:00- 2:45	Second Computer Session
2:45- 3:30	Final Class Session

find difficult and this interactive approach gives them immediate feedback and response. Rather than being thrown to the computer with no class discussion until, at earliest, the next day, they have their problems cleared up immediately. For the most part, they are willing to give up the better part of a day for this experience, and the increased level of learning is noticable.

3. Sessions Following M.E. Day

After Managerial Economics Day, a sequence of four hour and twenty minute-classes, spanning a two week period, are conducted on further aspects of linear programming fundamentals. The primary purpose of these classes are to reinforce the concepts which were introduced during Managerial Economics Day and to show how linear programming can be implemented in a variety of decision environments. The pedagogical format is as follows. We began the first of the four classes with a set of exercises in which the students are expected to successfully formulate four smaller linear programs, run them on CLP, and interpret the solution results. The students are then gradually introduced to more difficult formulations in the remaining three days. By the final class, they have become familiar with some of the basic issues involving model formulation and selection. During these four sessions a number of other issues arise, such as how to deal with uncertainty and the mechanism for pricing out the non-basic variables. The highlights of these four class sessions are described in the remainder of this section.

The first session consists of simple formulation exercises -- a machine shop problem, a transportation network, an additional marketing constraint on the Sherman Motors problem, and a simple bond portfolio investment in which the shadow prices are not particularly meaningful because the constraints take the form of ratios. A topic of note in the teaching of these exercises is the manner in which the student's participate in teaching as well as learning. Since interaction is a crucial aspect of the case method, the instructor may start the discussion by asking a non-quantitative student to show how he or she solved the first exercise. If there is little difficulty with this aspect, another non-quantitative student might be asked to qualitatively describe the formulation. Eventually the conversation gets around to discussion and defining the various basic types of constraints that can occur in a linear program:

- product (quality mix),
- capactiy or resource limits, and
- supply-use constraints (flow).

If there had been a severe problem with the initial formulation, the following pedagogical device has proven useful. The confused student is asked to concisely and verbally describe any single constraint while the instructor transcribes the verbal description onto the blackboard. At this point, the instructor can either show how the sentence can be easily parsed into an equation or ask for volunteers to do likewise. In this manner the poets are shown that almost anyone can formulate mathematically a linear program, provided a precise description of each restriction is developed. We are careful not to overwhelm the students with mathematical sophistication at the beginning, yet ensure that their logical reasoning is sound and rigorous.

The next class session depicts a simple capital investment decision problem, i.e., the Mitchell Enterprises case.* Here the problem is to formulate a linear programming model for investing in five projects (A, B, C, D, E) over a four year horizon with the objective of maximizing the accumulated value of the portfolio. An opportunity rate is not provided; however, a 6% short-term bank account is available for any money which is not invested in a given year. Exhibit V depicts the cash flows per dollar invested for each of these projects and years.

Exhibit V

CASH FLOW PER DOLLAR INVESTED

Project:	A	B	C	D	E
Year 1975	-1.00	0	-1.00	-1.00	0
1976	+.30	-1.00	+1.10	0	0
1977	+1.00	+.30	0	0	-1.00
1978	0	+1.00	0	+1.75	+1.40

With this information, the students are asked to formulate and evaluate a linear programming model for this problem. In addition to the case, they are given in advance a supplement containing the CLP solution to this problem. In the class discussion, the following important topics are usually covered:

---

*Intercollegiate Case Clearing House #4-176-160

1) a review of model formulation,
2) a demonstration that the opportunity rates can be derived from the optimal shadow prices, and
3) a proof that opportunity rates are not necessarily constant for each year.

In addition, a brief discussion of how to deal with uncertainties via sensitivity analysis takes place if time remains. This case has been well received because of its implications on traditional discounting methods - a firm's hurdle rate depends on the set of opportunities it has available. In addition, it is not a typical production scheduling situation — which many synonymize with linear programming models.

On the third day the Red Brand Canners* case is used. This case involves the production of canned tomatoes, tomato juice, and tomato paste. Information about fixed and variable costs, allocated overhead expenses, marginal profit, and so forth, is provided. The students are again asked to formulate the linear programming model and to defend their formulation in class. This session is noteworthy because the case forces students to extract the relevant accounting figures and to use this information within the model. It also vividly brings out the limiting assumptions of linear programming, i.e., constant returns to scale, continuous variables, non-negativity and so on. As far as pedagogy is concerned, the class discussion can be constructed to compare linear programming with "back of the envelope-seat of the pants" approach. To accomplish this comparison, there is an appealing, but incorrect, formulation which can be proposed by the instructor as the easiest way to solve this scheduling problem. A "better" student will counter by describing why the intuitive approach is incorrect and giving the correct formulation. Often, other students are unprepared to challenge this alternative formulation and the class anxiety rises until the logical fallacy is uncovered. The purpose of this subterfuge is demonstrating to the students that they must be prepared to defend their formulation to their superiors even though their superiors may not understand mathematical models or symbols. This important lesson is difficult to accomplish outside the case format.

In the final linear programming class session, the Okanagan Lumber Company* case is presented. This case involves the production of plywood paneling from various types of lumber. The intermediate steps in the plywood mill are described in detail. The problem of scheduling and allocating resources is depicted as a linear program. Since the model has approximately 50 constraints and 75 variables, the tableau is provided to the students. They are asked several sensitivity analysis questions and thereby develop an appreciation of the model. By this time, the students see that they can understand even a rather complex formulation if given enough time to sift through the details. Because of its complexity, there are many ways to teach this case. Perhaps the most important lesson to be learned is that the modeling process can uncover new avenues rather than just showing how to travel old paths. To demonstrate this idea, the instructor can utilize the shadow prices in conjunction with prevailing market prices for lumber to "invent" new product lines. In essence the students learn how to price-out non-basic variables - whenever a combination of marginal profits minus marginal costs is positive the product is profitable and worth introducing, which is equivalent to saying that the product's reduced cost is positive and can thereby enter the basis at a positive level. This case concludes the linear programming segment of the Managerial Economics course.

As a final observation, we compared the case approach with the usual lecture method. One of the authors taught linear programming to M.S. and Ph.D. students from Harvard's Division of Engineering and Applied Mathematics. This class followed a more traditional lecture format and the students generally had more extensive mathematical training (linear algebra, etc.) than MBA students. The same case given to MBA's on their final exam was given to these students as a part of their final examination. It is interesting to observe that the non-quantitative MBA's examination results compare favorably with the Ph.D. student's examination results. The "poets" were especially adept at putting the model in context with respect to the total investment decision - including how to deal with the underlying uncertainties.

4. Conclusions

We have attempted to show in this article how linear programming fundamentals can be successfully taught to a diverse audience of MBA students. A parallel situation exists within an executive management program at Harvard, in which highly experience managers return to academia for a 13 week retraining period. The linear programming aspects of this program are almost identical to those described in Sections 2 and 3 of this paper.

---

*Intercollegiate Case Clearing House #4-174-102.

*Intercollegiate Case Clearing House #3-176-638.

Although we feel that our pedagogical approach accomplished the goals which have been set forth, several important questions remain. First, does the successful application of sophisticated mathematical models require an appreciation by the decision maker of the solution strategies in the model? An airline pilot may not comprehend the physical laws of aerodynamics, but he can still fly. Nonetheless, a thorough understanding of the airplane can be an advantage to the pilot whenever something goes wrong. Perhaps this is why automobile race drivers are intimately familiar with the mecanical aspects of their machines.

We suspect that the pertinent answer to this question lies in the resolution of the tradeoff between intention and practicality. In other words, since there is limited time for learning, priorities must be established. The chosen mix of applications and theory depends upon how one resolves these issues.

As a second question, how do business schools deal with field study, the recent phenomenon which is occuring in many law schools? Here teams of students address real-world problems by going out into government or business. Because of the size of the MBA program at the Harvard Business School, it would be difficult to implement such a program; however, this topic will need to be further explored.

Thirdly, will the dramatic improvments in mini-computers, telecommunications, and self-paced learning techniques have a significant impact on how linear programming is taught? One of our associates has developed a cassette-based system* for decision analysis, and similar ideas could be easily exploited in linear programming. Several business schools are already engaged in developing self-paced instructional materials.

REFERENCES

1. Ashenhurst, R.L. (ed.), "A Report of the ACM Curriculum Committee on Computer Education for Management," Communications of the ACM, 15, 5, (May 1975) pp. 363-398.
2. Churchman, C. West (ed.), "Educational Issues in the Management Sciences and Operations Research," Management Science, 17, 2, (October 1970) pp. B1-B53.
3. Dickson, G.W. and V.T. Dock, "Graduate Professional Programs in Information Systems: A Survey," Interfaces, 6, 1, (November 1975) pp. 38-43.
4. Healy, D.F., "Education - the Critical Link in Getting Managers to Use Management Science," Interfaces, 2, 3, (May 1972).
5. "POM/ME Workshop," Intercollegiate Case Clearinghouse, #9-676-135, March 16-19, 1976.
6. Wagner, H.M., "The ABC's of OR," Operations Research, 19, 6, (October 1971), pp. 1259-1281.

---

*This system was developed by Howard Raiffa. For more information, see Analysis for Decision-Making, Encyclopedia Britannica, 1974.

"Experiments with Computer Aided Self-Paced Instruction for Mathematical Programming Education"

A. Ravindran
School of Industrial Engineering
Purdue University, West Lafayette
Indiana 47907

Arthur Sinensky

New York, New York 10000

Thomas Ho
Department of Computer Science
Purdue University, West Lafayette
Indiana 47907

ABSTRACT

In this paper the authors discuss their experiences in converting a lecture-oriented mathematical programming course to the Self-Paced Instructional (PSI) format. This is an elective course for students from all the departments (Engineering, Science, and Management). We will discuss the different strategies, including the development of conversational computer programs, which were employed to implement the PSI concept in this course so that its educational objectives are fully met. We will also discuss our experiences with the PSI system, its pros and cons, and the students' response to self-paced learning.

Introduction

Undergraduate engineering curricula have become much more flexible during the past half-dozen or so years. Students are able to obtain a measure of specialization in one or two areas within a specified engineering discipline; for example, in the School of Industrial Engineering at Purdue the undergraduate students can specialize in Operations Research, Systems Engineering, Human Factors, Management, and Manufacturing Processes. Most of this flexibility has been obtained by increasing the number of elective courses allowed in the curriculum without increasing the credit hours required for graduation.

The author has been teaching for the past seven years a course on Mathematical Programming for undergraduate and graduate students. This is an elective course for all industrial engineering students. In addition to students from other branches of engineering, the course attracts students from industrial management, agricultural economics, mathematics, computer science, and statistics. Because of this, the composition of the class and the interests of the students vary widely. The course is offered three times a year and attracts 150-200 students.

The basic objective of this course is to introduce the students to mathematical modeling (in particular linear programming models) for solving real-world problems. The major portion of the course deals with solution techniques for these models. From past experience in teaching this course, it has been observed that the non-engineering students prefer to see more emphasis on linear programming theory and computational methods. In contrast, the engineering and business students want more emphasis on case studies describing numerous applications of linear programming. They even express varied interests in the linear programming topics to be discussed in class. For instance, the mathematics students want the inclusion of advanced topics like game theory, and decomposition methods. The computer science students like to see the computer implementation of linear programming algorithms. The civil engineers show more interest in topics like network analysis, transportation problems, and critical path methods. This poses enormous problems in structuring and teaching this course. In previous years, a middle-of-the-road approach was taken, so that it would hopefully satisfy most of the students' interests.

Use of Personalized Self-Paced Format

Traditionally all students enrolled in a specific course progress through the material defined by the syllabus at a given rate. Concepts presented in a textbook are supplemented with regularly scheduled lectures by the course instructor, homework problems are turned in at the proper time, lab experiments are performed during scheduled hours, and hour tests are taken in class size groups when the instructor schedules them. Fortunately this is not a rut in which the innovative teacher must remain. Through the years we have seen the growth in use of concepts of indiviualized, self-paced instruction and open laboratories (see References [1], [2], [3], and [4]). The American Society for Engineering Education has been concerned with effective college teaching for a number of years and usually devotes the March issue of Engineering Education each year to articles concerned with innovative methods of teaching in engineering curricula. A recent survey by Moodie [5] reports various innovative teaching methods used in industrial engineering courses across the country. However, we often find that the major difference between the self-paced version of a course and its lecture counterpart is merely the self-pacing; there is still a strict list of topics to be covered. Variation within the listed course content is only offered to the student through the vehicle of writing a term paper on a special topic of his interest, not included in the syllabus.

The author of this paper felt the need to expand the amount of flexibility offered to a student enrolled in the mathematical programming course beyond that which is described in the preceding paragraphs. Because of the diversified interests found among the students, it was felt that a new Personalized Self-Paced Instructional (PSI) system could best serve the students' needs. The new PSI system developed by the primary author [7] combines the traditional lectures, self-paced and mastery learning to provide maximum flexibility. To obtain any degree of course topic flexibility it is necessary to evaluate each course in terms of its educational objectives. What is the terminal behavior we desire of a student when he completes this course? Then, in order to make additional, optional material available to students enrolled in this course, it is necessary to determine the minimum, necessary requirements from the original course content. In this way specific topics are made required content of this course. After this is completed, the additional (optional) material is made available to the student in the form of elective modules.

Thus, in the new system, the subject matter for this course is divided into some basic units, and some optional units. Class lectures are given for the basic units only. The students elect the optional units they want to learn according to their interests, and prepare for them on their own time (off class hours). For this reason, one-third of the scheduled class hours are cancelled to provide time for independent study.

BASIC UNITS: The basic units are so chosen that they provide the students a minimal amount of knowledge in the fundamentals of linear programming. These include:

Unit 1 - Formulation of Linear Programs: To construct linear programming models of real world problems.

Unit 2 - The Simplex Method: To learn how systems of linear equations are solved; to understand the basic mathematical principles underlying the simplex algorithm; to use the tableau form of the simplex method to solve small problems; and to use the computer code for solving large linear programs.

Unit 3 - Duality Theory and its Applications: Symmetric and asymmetric duals, economic interpretation, duality theorems and applications, shadow prices, and the dual simplex method.

Unit 4 - Sensitivity Analysis: Variation of cost coefficients, changing RHS constants, changing constraint matrix, adding new constraints and parametric programming with respect to cost and RHS vectors.

OPTIONAL UNITS: The optional units are selected to provide diversification, and meet the varied and special interests of the students. For instance, during the summers of '74, '75, and '76, when the PSI system was tried for this course, the following optional units were offered to the students:

Unit 5 - Project Networks and PERT/CPM

Unit 6 - Transportation and Assignment Problems

Unit 7 - Variants of the Simplex Method

Unit 8 - Nonlinear Optimization

Unit 9 - Bounded Variable Linear Program

Unit 10 - Bimatrix Games

Unit 11 - Integer Programming

In addition, an individual project is always made available as an optional unit. For each of the units (basic and optional) the students are provided with study guides. Russell [8] calls these "modular materials" and provides an excellent guide to the design, selection, utilization, and evaluation of these study guides. Basically, the study guides contain the objectives of the unit, topical outline, reading materials, and sample problems. Examinations on various units are given throughout the semester. Though excellence is required to pass a unit, the students can repeat the unit examinations any number of times without penalty. For a minimal passing grade in the course, the students have to complete all the basic units. Assignment of higher grades is a function of the number of optional units they pass.

Computer Aids in the PSI System

From the initial experience in teaching this course in PSI format, it was found that considerable time was spent with the students, by both the professor and the teaching assistants, in helping them learn the optional units for which there were no formal lectures. This increased the manpower needs of this course. It was felt that this could be reduced to a great extent by using the computer as a teaching aid for self-paced learning. The computer-aided teaching can help them learn the various linear programming techniques and their applications on their own time (on/off class hours), and at their own pace.

With the help of a teaching grant from the Purdue Parents' Association, interactive computer programs have been developed in such a way that they will illustrate and test the students' understanding of the subject matter. The students solve problems illustrating various linear programming algorithms in a conversational mode by answering a series of questions. The programs are designed so that the computer does all the time consuming arithmetic calculations, while the "thinking" is done by the students. The student essentially directs the computer to calculate whatever quantities are needed for solving the problem by a given algorithm. Based on these, he/she instructs the computer to proceed step by step in the required manner to arrive at the optimal solution to the problem. We have provided in the program a verification routine so that all the steps and instructions given by the students are checked by the computer for correctness.

The arithmetic calculations in linear programming techniques generally consume more than 90% of the total time in solving a problem. Since this time is eliminated by the computer, the students are able to use their study time more effectively. They can solve a variety of problems to test their understanding. A series of illustrative problems emphasizing various concepts and practical difficulties encountered in solving linear programs are also available to the students. Thus a student gets a better and a more complete exposure to the subject matter itself. At present the interactive programs are used to reinforce methodology and to interpret results. The unit examinations do not utilize these interactive programs.

Description of the Interactive Computer Program

Conversational Features

The program is completely conversational. All input and output is handled via remote terminals. The system always directs the user to the next step of the simplex algorithm by asking a question related to some aspect of the algorithm. At all times, the user's response is checked for correctness. If the user should respond incorrectly to one of the system's queries, an immediate feedback will be sent to the user with an explanation of why the response is incorrect and how to go about finding the correct answer.

Error Detection for Inappropriate Student Responses

Each student response to a system question is checked for correctness. If the student's answer is correct, the next question is asked by the system. However, if the response is not correct, feedback explaining the error is given at the terminal. In some cases, the same question is reasked and the student is given further opportunity to respond correctly. In other instances, the system supplies the correct answer and then proceeds to the next question.

Error checking is accomplished by having the system solve the given linear program step-by-step as the user solves the same problem. For instance, before asking the user to supply the index of the pivot row in the simplex algorithm, the system will have determined that information. Then the user will be asked to supply the same. After the user enters the response, it is a simple matter to check the response against the result already determined by the system. If the user's response is correct, the next question is printed on the terminal. If the response is incorrect, a message to that effect is printed with a brief explanation of why the response is not correct.

For instance, on line A (pg. 5) of the enclosed example in the Appendix, the student is asked, "IS THE PROBLEM IN STANDARD FORM?". In this case, the problem is not in standard form. But the student has responded, "YES". Thus, the feedback given is "INCORRECT RESPONSE - THE PROBLEM IS NOT IN STANDARD FORM, NOT ALL OF THE CONSTRAINTS ARE EQUALITIES". Then the next question is generated.

User Assistance

In addition to the brief explanations provided when the user gives an incorrect response, another form of explanatory assistance is offered by the system. At several points, the student is asked to name the next step of the algorithm. The possible responses and their meanings are:

RELPROFIT -	print the relative profit vector
RATIOS -	calculate ratios of right-hand sides divided by the corresponding entry of the pivot column
PIVOT -	perform a pivot operation

In any instance where one of the above responses is the correct one, the user may type "HELP" if he is uncertain of which choice is the correct one. After "HELP" is typed, a message indicating which is the correct choice and why it is correct will be provided. A possible extension would be to allow the user to ask for "HELP" at any point during the execution of the program. This should be included in the next version of the system.

An example of the use of this command is given in line B (pg. 5) of the accompanying example. The question asked is, "WHAT IS THE NEXT STEP?". In this case, the correct response would be "RELPROFIT", since the user must look at the relative profit vector. The user has typed "HELP" and thus receives an appropriate explanation.

Implementation Description

Programming Language Requirements

The system was programmed in FORTRAN-IV on the CDC 6500. A language with better character string handling facilities would have been a better choice. However, the constraining factor was the interactive field length restriction on the Purdue system. Interactive jobs are limited to a maximum field length of 21K. The object code generated by FORTRAN was able to fit into this space and still allow some room for extensions. On an IBM system with time-sharing, APL would be a better choice.

Algorithmic Description

At present three mathematical programming algorithms have been programmed in the interactive system. These include a primal simplex and a dual simplex algorithm to solve linear programs, and a branch and bound algorithm for solving integer linear programming problems. Each program's logic is based on the description of that algorithm given in the text by Phillips, Ravindran, and Solberg [6]. Students are asked to input problems of their own choosing. This allows for great flexibility in the instructional problems chosen rather than limiting the selection to just a few. The instructor may also suggest some problems which demonstrate the basics of the algorithm. After mastering the basics, the students can experiment with problems of their own choosing.

Concluding Remarks

The PSI system is currently being well received by the students. The authors have evaluated the courses by several measures: comparison with test results in other years, student election of more advanced courses in these areas, student acceptance as indicated on written course critiques, etc. In all cases, the self-paced method of instruction was judged in general to be superior for this particular course. (See Table 1 for student comments.) This is not to say that the usual problems of some students falling behind schedule did not exist, because they did. Effort is expended during the semester to help student motivation. Also, the PSI system consumes more faculty time initially in the planning of the units, and preparation of study guides and tests.

Considerable time is still being spent on an individual basis with some students in helping them learn the optional units for which there are no formal lectures. To offset this the author has obtained another research grant to support the preparation of video cassette tapes for the optional units. The School of Industrial Engineering has recently acquired a TV monitor and a cassette playback unit. These video tapes can be played on these units by the students at their own time. We hope to have these tapes prepared by the end of the year for students' use.

Acknowledgements

The support for developing the interactive computer programs came from an innovative teaching grant awarded to Professor Ravindran from the Purdue Parents' Association. Preparation of the video cassette tapes for TV viewing will be supported by a faculty grant awarded to Professor Ravindran for Instructional Development and Innovation.

References

[1] Block, J. H., Mastery Learning, Holt, Rinehart, and Winston,

[2] Dell, H. F., Individualizing Instruction, Science Research Associates, Palo Alto, California, 1972.

[3] Esbensen, T., Working with Individualized Instruction, Fearon, Palo Alto, California, 1968.

[4] Keller, F. S., "Good Bye Teacher ...", Journal of Applied Behavior Analysis, No. 1, Spring 1968.

[5] Moodie, C. L., "Report on a Survey of Innovative Teaching Methods Used in Industrial Engineering Education", Research Memorandum No. 73-8, School of Industrial Engineering, Purdue University, West Lafayette, Indiana, July 1973.

[6] Phillips, D. T., A. Ravindran, and J. Solberg, Operations Research: Principles & Practice, John Wiley & Sons, New York, 1976.

[7] Ravindran, A. and C. L. Moodie, "Increased Course Flexibility Through Self-Paced Instruction", Proceedings of the Frontiers in Education Conference (ASEE), Atlanta, Georgia, October 1975, (pp. 400-403).

[8] Russell, J. D., Modular Instruction, Burgess Publishing Co., Minneapolis, Minnesota, 1974.

APPENDIX

The solution to the following linear program by the interactive simplex algorithm is shown as an illustration:

$$\begin{array}{ll} \text{Maximize} & z = 10x_1 + 6x_2 + 4x_3 \\ \text{Subject to} & x_1 + x_2 + x_3 \le 100 \\ & 10x_1 + 4x_2 + 5x_3 \le 600 \\ & 2x_1 + 2x_2 + 6x_3 \le 300 \\ & x_1, x_2, x_3 \ge 0 \end{array}$$

```
LINEAR PROGRAMMING SELF-PACED INSTRUCTION

ENTER THE NUMBER OF CONSTRAINTS FOLLOWED BY
A BLANK FOLLOWED BY THE NUMBER OF VARIABLES

: 3 3

ENTER THE LINEAR PROGRAM

IS THE OBJECTIVE FUNCTION TO BE MINIMIZED OR
MAXIMIZED?

: MAX

ENTER THE C VECTOR

: 10 6 4

ENTER THE A MATRIX, ROWWISE

: 1 1 1
: 10 4 5
: 2 2 6

ENTER THE B VECTOR

: 100 600 300

ENTER THE CONSTRAINT TYPES:
   < FOR LESS THAN OR EQUAL TO,
   > FOR GREATER THAN OR EQUAL TO,
   = FOR EQUALITY CONSTRAINTS

: < < <

(A) IS THE PROBLEM IN STANDARD FORM?

: YES

INCORRECT RESPONSE -
THE PROBLEM IS NOT IN STANDARD FORM


NOT ALL OF THE CONSTRAINTS ARE EQUALITIES


CHECK THE CONSTRAINTS TO SEE IF THERE ARE ANY
NEGATIVE RIGHT-HAND SIDES.  ALL SUCH CON-
STRAINTS NEED TO BE MULTIPLIED BY -1.
WHICH CONSTRAINTS NEED TO BE MULTIPLIED BY
-1?
TYPE 0 (ZERO) IF NONE

: 0

FOR EACH CONSTRAINT, TYPE A + 1, -1, OR 0
DEPENDING ON WHETHER A SLACK VARIABLE IS
NEEDED, A SURPLUS VARIABLE IS NEEDED, OR
NEITHER ONE IS NEEDED

: 1 1 1

THE SLACK AND SURPLUS VARIABLES HAVE BEEN
INSERTED

ARE ARTIFICIAL VARIABLES NEEDED?

: YES
```

```
INCORRECT RESPONSE -
NO ARTIFICIAL VARIABLES ARE NEEDED BECAUSE
THERE EXISTS AN INITIAL FEASIBLE BASIS

THE CURRENT TABLEAU IS:

COST     10      6      4      0      0      0
         X1     X2     X3     X4     X5     X6      B
BASIS
X4      1.00   1.00   1.00   1.00    .00    .00100.00
X5     10.00   4.00   5.00    .00   1.00    .00600.00
X6      2.00   2.00   6.00    .00    .00   1.0 300.00

WHAT ARE THE VALUES OF THE BASIC VARIABLES
IN THE ORDER IN WHICH THEY APPEAR IN THE
TABLEAU

: 100 600 300

WHAT IS C(B), THE VECTOR OF COST COEFFICIENTS
OF THE BASIC VARIABLES IN CORRESPONDING ORDER?

: 0 0 0

THE VALUE OF THE OBJECTIVE FUNCTION =       .00

(B) WHAT IS THE NEXT STEP?
    CHOOSE FROM PIVOT, RATIOS, OR RELPROFIT
    IF YOU DONT KNOW, TYPE HELP

: HELP


THE NEXT STEP OF THE SIMPLEX ALGORITHM  IS TO
LOOK AT THE RELATIVE PROFIT VECTOR AND
DETERMINE WHETHER THE CURRENT BASIC FEASIBLE
SOLUTION IS OPTIMAL.  DETERMINE THE PIVOT
COLUMN AND HENCE THE VARIABLE WHICH ENTERS
THE BASIS BY CHOOSING THE VARIABLE WITH THE
LARGEST RELATIVE PROFIT COEFFICIENT.  IN
ORDER TO SEE THE RELATIVE PROFIT VECTOR, TYPE
RELPROFIT.


WHAT IS THE NEXT STEP?
CHOOSE FROM PIVOT, RATIOS, OR RELPROFIT
IF YOU DONT KNOW, TYPE HELP

: RELPROFIT

THE RELATIVE PROFIT VECTOR IS:

10.00  6.00  4.00  .00  .00  .00

IS THE TABLEAU OPTIMAL?

: YES

INCORRECT RESPONSE -
FOR A MAXIMIZATION PROBLEM, THE TABLEAU IS
OPTIMAL WHEN ALL THE RELATIVE PROFIT COEF-
FICIENTS ARE NON-POSITIVE

SELECT THE PIVOT COLUMN

: 1

WHAT IS THE NEXT STEP?
CHOOSE FROM PIVOT, RATIOS, OR RELPROFIT
IF YOU DONT KNOW, TYPE HELP
```

```
THE NEXT STEP OF THE SIMPLEX ALGORITHM
IS TO DETERMINE THE PIVOT ROW AND HENCE
THE VARIABLE WHICH LEAVES THE BASIS USE THE
MINIMUM RATIO RULE TO DETERMINE THE BASIC
VARIABLE TO LEAVE THE BASIS. TO CALCULATE
RATIOS TYPE RATIOS


: RATIOS

FOR EACH ROW, TYPE A 1 IF A RATIO NEEDS TO
BE CALCULATED OTHERWISE TYPE A 0

: 0 0 0

THE RESPONSE FOR ROW 1 IS INCORRECT


THE RESPONSE FOR ROW 2 IS INCORRECT


THE RESPONSE FOR ROW 3 IS INCORRECT

THE CORRECT RESPONSE IS:    1   1   1

THE RATIOS ARE
ROW NO.        RATIO

   1          100.00
   2           60.00
   3          150.00

CAN A PIVOT OPERATION BE PERFORMED?

: NO

INCORRECT RESPONSE -
A PIVOT OPERATION CAN BE PERFORMED SINCE AT
LEAST ONE RATIO HAS BEEN CALCULATED


SELECT THE PIVOT ROW

: 3

INCORRECT RESPONSE -
THE ROW WITH THE SMALLEST RATIO IS THE PIVOT
ROW

SELECT THE PIVOT ROW

: 2

WHAT IS THE NEXT STEP?
CHOOSE FROM PIVOT, RATIOS, OR RELPROFIT
IF YOU DONT KNOW, TYPE HELP

: HELP

NOW THAT THE PIVOT ELEMENT HAS BEEN
DETERMINED, A PIVOT OPERATION CAN BE
PERFORMED.  TYPE PIVOT TO PERFORM THE
PIVOT OPERATION


WHAT IS THE NEXT STEP?
CHOOSE FROM PIVOT, RATIOS, OR RELPROFIT
IF YOU DONT KNOW, TYPE HELP
```

```
: PIVOT

 THE CURRENT TABLEAU IS:

COST  10     6     4     0     0     0
      X1    X2    X3    X4    X5    X6        B
BASIS
X4   .00   .60   .50  1.00  -.10   .00    40.00
X1  1.00   .40   .50   .00   .10   .00    60.00
X6   .00  1.20  5.00   .00  -.20  1.00   180.00

 WHAT ARE THE VALUES OF THE BASIC VARIABLES
 IN THE ORDER IN WHICH THEY APPEAR IN THE
 TABLEAU

: 40 60 180

 WHAT IS C(B), THE VECTOR OF COST COEFFICIENTS
 OF THE BASIC VARIABLES IN CORRESPONDING ORDER?

: 0 10 0

 THE VALUE OF THE OBJECTIVE FUNCTION =  600.00

 WHAT IS THE NEXT STEP?
 CHOOSE FROM PIVOT, RATIOS, OR RELPROFIT
 IF YOU DONT KNOW, TYPE HELP

: RELPROFIT

 THE RELATIVE PROFIT VECTOR IS:

    .00   2.00   -1.00    .00   -1.00   .00

 IS THE TABLEAU OPTIMAL?

: NO

 SELECT THE PIVOT COLUMN

: 2

 WHAT IS THE NEXT STEP?
 CHOOSE FROM PIVOT, RATIOS, OR RELPROFIT
 IF YOU DONT KNOW, TYPE HELP

: RATIOS

 FOR EACH ROW, TYPE A 1 IF A RATIO NEEDS TO BE
 CALCULATED OTHERWISE TYPE A 0

: 1 1 1

 THE RATIOS ARE
 ROW NO.        RATIO

    1           66.67
    2          150.00
    3          150.00

 CAN A PIVOT OPERATION BE PERFORMED?

: YES

 SELECT THE PIVOT ROW

: 1

 WHAT IS THE NEXT STEP?
 CHOOSE FROM PIVOT, RATIOS, OR RELPROFIT
 IF YOU DONT KNOW, TYPE HELP
```

```
: PIVOT

 THE CURRENT TABLEAU IS:

COST   10     6     4     0           0
       X1    X2    X3    X4     X5    X6    B
BASIS
X2    .00  1.00   .88  1.67   -.17   .0 66.67
X1   1.00   .00   .17  -.67    .17   .0 33.33
X6    .00   .00  4.00 -2.00   -.00  1.0 100.00

 WHAT ARE THE VALUES OF THE BASIC VARIABLES
 IN THE ORDER IN WHICH THEY APPEAR IN THE
 TABLEAU

: 66.67  33.33  100.00

 WHAT IS C(B), THE VECTOR OF COST COEFFICIENTS
 OF THE BASIC VARIABLES IN CORRESPONDING ORDER?

: 6  10  0

 THE VALUE OF THE OBJECTIVE FUNCTION =  733.33

 WHAT IS THE NEXT STEP?
 CHOOSE FROM PIVOT, RATIOS, OR RELPROFIT
 IF YOU DONT KNOW, TYPE HELP

: RELPROFIT

 THE RELATIVE PROFIT VECTOR IS:
    .00   .00   -2.67   -3.33   -.67   .00

 IS THE TABLEAU OPTIMAL?

: YES


 DO YOU WANT TO BEGIN A NEW PROBLEM?

: NO
+++LOG

 TOB L024 11.44.51. 10/07/75.
 ESTIMATED SESSION COST $    .17
 PLEASE TURN OFF TERMINAL. TNX.
```

304

Table 1. Results of Student Survey

	Summer 1974			Summer 1975			Summer 1976		
1. No. of Students Enrolled	25			44			49		
2. No. of Students Responded	23			36			39		
3. Reaction to the PSI System	Yes	No	Neither	Yes	No	Neither	Yes	No	Neither
a. Favorable initial response (semester beginning)	23	6	4	25	8	3	28	9	2
b. Favorable final response (semester end)	18	5	0	33	2	1	32	6	1
c. Helped me to achieve desired goals	19	3	1	35	1	0	30	8	1
d. Able to plan my studies better	17	5	1	31	5	0	29	9	1
e. Helped me to get a better grade	10	8	5	22	7	7	27	10	2
f. Would have preferred traditional format	5	18	0	1	32	3	6	30	3
g. Learned more under PSI	10	7	6	24	5	7	24	11	4
h. Worked harder under PSI	8	13	2	18	15	3	14	21	4
i. Worked less under PSI	12	9	2	11	21	4	14	24	1

IMPORTANCE OF MODELLING
FOR INTERPRETATION OF
LINEAR PROGRAMMING PROBLEMS

Leonard W. Swanson
Northwestern University

Abstract

The construction of the mathematical model for a linear programming problem requires extreme care in order that it be effective. If one uses as few variables as possible, he may find that it is efficient in computation time but ineffective for sensitivity analysis. This paper uses a relatively uncomplicated example to show the way in which proper modelling enables one to extract important analyses not obtainable through a simpler model. The approach has been successfully used in teaching linear programming, particularly for explaining the concept of duality. The same methods can be highly effective in managerial applications.

Introduction

It is often thought that one is being very efficient if he solves a Linear Programming Problem by using the fewest possible number of variables. If one is interested only in the solution and not in any kind of sensitivity analysis, this may be true, but if one is interested in the effect of a variation of the parameters of the problem, many more details may be required to enhance the analysis and interpretation.

Scheduling Problem

In order to consider the benefits of efficient modelling, consider the production scheduling problem shown in Figure 1 (no claim is made for the originality of the example).

A plant makes two products. A and B, which are routed through four processing centers 1, 2, 3, and 4 as shown by the solid lines in the enclosed diagram. If there is spare capacity in center 3, it is possible to route product A through 3 (dotted line) instead of going through 2 twice, but this is more expensive.

Given the information in Table 1 and Table 2, how should production be scheduled so as to maximize profit? (By production schedule is meant

PRODUCTION SCHEDULING

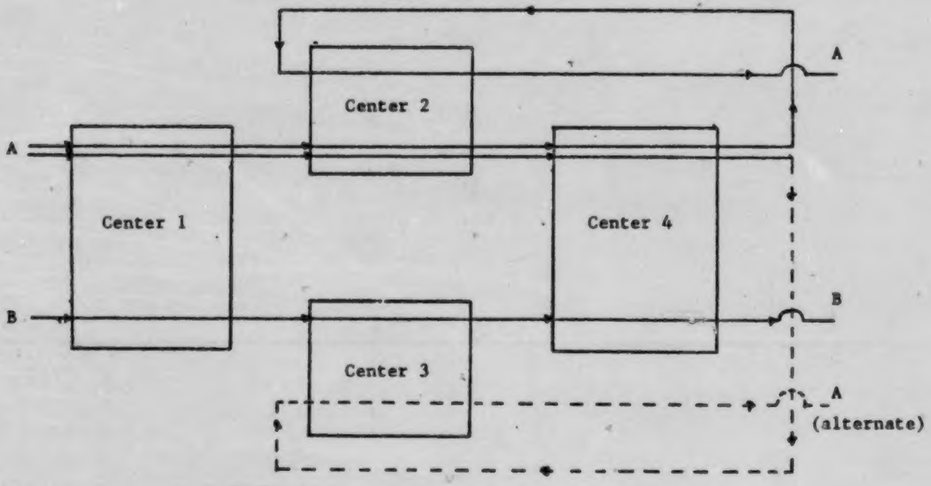


Figure 1

the specification of the following three amounts:

(1) The daily amount of raw material used for A, regular route,

(2) The daily amount of raw material used for A, optional route,

(3) The daily amount of raw material used for B.

Assume that sufficient storage capacity is available at no additional cost.)

Table 1

	Center	Inputs, gals. per hour	% recovery	Running cost per hr. $
A	1	300	90	150
	2(1st pass)	450	95	200
	4	250	85	180
	2(2nd pass)	400	80	220
	3	350	75	250
B	1	500	90	300
	3	480	85	250
	4	400	80	240

Table 2

Product	Raw Material cost per gal.	Sales price per finished gal.	Maximum daily sales, gal. of finished product
A	5	20	1700
B	6	18	1500

Centers 1 and 4 run up to 16 hours a day; centers 2 and 3 run up to 12 hours a day. A final restriction is furnished by shipping facilities, which limit the daily output of A and B to a total of 2500 gallons.

Consider first the formulation of the problem in a form in which as few variables as possible are used. Therefore

Let x_1 = the number of gallons of input of A, which is to follow the regular processing route

x_2 = the number of gallons of input of A, which is to follow the alternate processing route

x_3 = the number of gallons of input of B, which has only one route.

Figure 2 is a representation of the problem incorporating these variables and also the information from Tables 1 and 2.

On each path through a particular center, there are two numbers; the left hand number is the number of gallons per hour that can be processed and the right hand number is the cost per hour for processing.

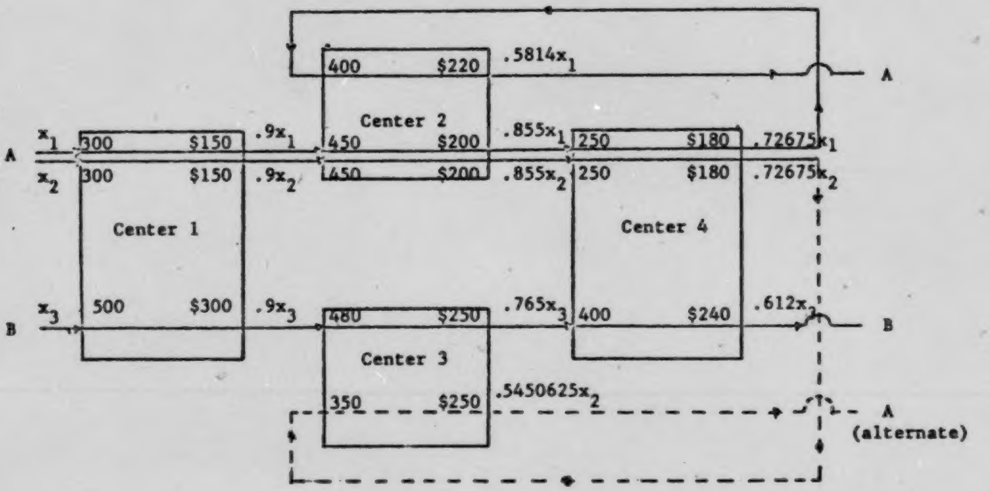
Mathematical Statement of the Problem

Find $x_1, x_2, x_3 \geqq 0$

Such that

Center capacities

$$\begin{cases} 5x_1 + 5x_2 + 3x_3 \leqq 24{,}000 \\ 13.74075x_1 + 7.2x_2 \leqq 43{,}200 \\ 34.884x_2 + 31.5x_3 \leqq 201{,}600 \\ 6.84x_1 + 6.84x_2 + 3.825x_3 \leqq 32{,}000 \end{cases}$$



A
B
x_1
x_2
x_3
300 $150
300 $150
500 $300
Center 1
$.9x_1$
$.9x_2$
$.9x_3$
400 $220
Center 2
450 $200
450 $200
$.5814x_1$
$.855x_1$
$.855x_2$
250 $180
250 $180
Center 4
400 $240
$.72675x_1$
$.72675x_2$
$.765x_3$
$.612x_3$
480 $250
Center 3
350 $250
$.5450625x_2$
A
B
A (alternate)

Figure 2

Sales Limits
$$\left[\begin{array}{l} .5814x_1 + .5450625x_2 \leq 1700 \\ \qquad\qquad .612x_3 \leq 1500 \end{array}\right.$$

Dock Capacity
$$\left[\, .5814x_1 + .5450625x_2 + .612x_3 \leq 2500\right.$$

and such that

$$4.7162875x_1 + 3.866543x_2 + 3.48825x_3$$

is a maximum.

Optimal Solution

In this form of the problem, one can get the optimum solution for carrying out the production schedule. The solution reveals that

1. 2923.98 gallons follow the regular route A but none follow the alternate route.
2. 1307.19 gallons follow route B.
3. The profit is $18,399.59.
4. The actual output of either A or B is not revealed directly although it can be obtained by additional simple computation.
5. Through the dual solutions one sees that the sales restriction on A is a limitation on profit and that Dock Capacity is also a limitation on profit. The per unit gains made by relaxing these constraints are given by the dual solutions.
6. Nothing in the solution is directly revealed concerning the effect of changing either the rates of gallon throughput for each of the centers nor the hourly costs of processing in each of the centers.
7. A study of the effect of changing the coefficients of the objective function is not very revealing since the coefficients in this function are a conglomeration of a number of incomes and costs.

This form of the model seems to be relatively neat and one might even pride himself in the brevity of the format thinking that restricting the problem to three variables and seven constraints is a minor triumph.

It would appear that a manager interested in producing the most effective schedule would require a sensitivity analysis which would study the effects of changes in processing rates and processing costs for each of the centers. Accordingly, in what follows I develop a model of the same problem which will enable the manager to learn a vast amount more about the operation and which would enable him to do a better job of effective management.

Alternate Statement of the Problem

As shown in Figure 3, x_1 and x_9 are used to represent the raw material inputs for A and B respectively. These variables are modified by recovery rates and new definitions are made so that variables x_1 to x_{12} designate various inputs and outputs in the processing stream. From Figure it can be seen that a solution for x_7 gives the final output of A, regular; a solution for x_8 gives the final output of A, alternate; and a solution for x_{12} gives the final output of B. Furthermore, the definition of these variables serve as constraints which involve the recovery rates. The definition of variables x_{13} through x_{20} serve as constraints which define the hourly processing rates of the centers. The recovery and hourly processing definitions and corresponding constraints are given below.

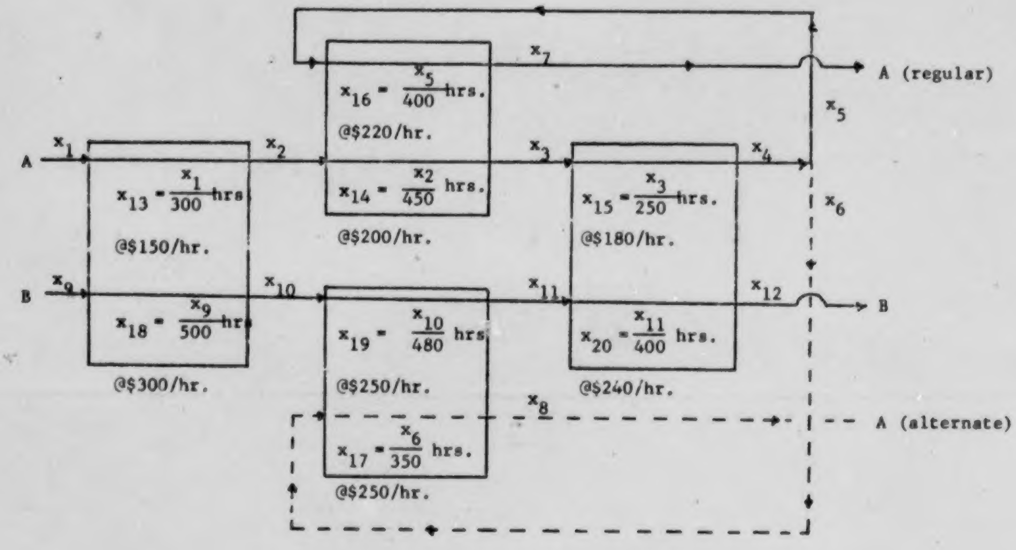


Figure 3

Recovery definitions:	When written as constraints:
$x_2 = .9x_1$	$x_2 - .90x_1 = 0$
$x_3 = .95x_2$	$x_3 - .95x_2 = 0$
$x_4 = .85x_3$	$x_4 - .85x_3 = 0$
$x_7 = .80x_5$	$x_7 - .80x_5 = 0$
$x_8 = .75x_6$	$x_8 - .75x_6 = 0$
$x_{10} = .90x_9$	$x_{10} - .90x_9 = 0$
$x_{11} = .85x_{10}$	$x_{11} - .85x_{10} = 0$
$x_{12} = .80x_{11}$	$x_{12} - .80x_{11} = 0$

Center Processing times:	When written as constraints:
$x_{13} = x_1/300$	$x_1 - 300x_{13} = 0$
$x_{14} = x_2/450$	$x_2 - 450x_{14} = 0$
$x_{15} = x_3/250$	$x_3 - 250x_{15} = 0$
$x_{16} = x_5/400$	$x_5 - 400x_{16} = 0$
$x_{17} = x_6/350$	$x_6 - 350x_{17} = 0$
$x_{18} = x_9/500$	$x_9 - 500x_{18} = 0$
$x_{19} = x_{10}/480$	$x_{10} - 480x_{19} = 0$
$x_{20} = x_{11}/400$	$x_{11} - 400x_{20} = 0$

We add the constraint which shows the balance for A regular and A alternate route as

$$x_4 = x_5 + x_6$$

If we then add the seven constraints of our smaller model in terms of the above variables we have a model with twenty-four constraints and twenty variables. The problem is then stated as follows:

Find x_1 to $x_{20} \geq 0$

such that

Recovery Definitions	$x_2 - .90x_1 = 0$
	$x_3 - .95x_2 = 0$
	$x_4 - .85x_3 = 0$
	$x_7 - .80x_5 = 0$
	$x_8 - .75x_6 = 0$
	$x_{10} - .90x_9 = 0$
	$x_{11} - .85x_{10} = 0$
	$x_{12} - .80x_{11} = 0$
Hourly Processing Rates	$x_1 - 300x_{13} = 0$
	$x_2 - 450x_{14} = 0$
	$x_3 - 250x_{15} = 0$
	$x_5 - 400x_{16} = 0$
	$x_6 - 350x_{17} = 0$
	$x_9 - 500x_{18} = 0$
	$x_{10} - 480x_{19} = 0$
	$x_{11} - 400x_{20} = 0$
Balance	$x_4 - x_5 - x_6 = 0$
Center Capacities	$x_{13} + x_{18} \leq 16$
	$x_{14} + x_{16} \leq 12$
	$x_{17} + x_{19} \leq 12$
	$x_{15} + x_{20} \leq 16$
Sales Constraints	$x_7 + x_8 \leq 1700$
	$x_{12} \leq 1500$
Dock Capacity	$x_7 + x_8 + x_{12} \leq 2500$

Such that

$$-5x_1 + 20x_7 + 20x_8 - 6x_9 + 18x_{12} - 150x_{13} - 200x_{14} - 180x_{15} - 220x_{16} - 250x_{17} - 300x_{18} - 250x_{19} - 240x_{20}$$

is a MAXIMUM.

Scheduling Parameters

The actual parameters for this problem in either form are:

1. Sales Limits on A and B
2. Dock Capacity
3. Hourly Capacities for centers 1, 2, 3, and 4.
4. Cost of Raw Material Input A and B
5. Income from Sales of A and B
6. Center 1

   $\left.\begin{array}{l}\text{Rate of processing (Gals/hour)}\\ \text{Cost of processing (\$/hour)}\end{array}\right\}$ for A

   Same as above for B
7. Center 2

   $\left.\begin{array}{l}\text{Rate of processing (Gals/hour)}\\ \text{Cost of processing (\$/hour)}\end{array}\right\}$ for A 1st pass

   Same as above — for A 2nd pass
8. Center 3

   $\left.\begin{array}{l}\text{Rate of processing (Gals/hour)}\\ \text{Cost of processing (\$/hour)}\end{array}\right\}$ for B

   Same as above for A alternate route
9. Center 4

   $\left.\begin{array}{l}\text{Rate of processing (Gals/hour)}\\ \text{Cost of processing (\$/hour)}\end{array}\right\}$ for A

   Same as above for B

Advantages of the Second Model

When using the short form of the problem, we are able to make a sensitivity study for items 1, 2, and 3 only. In the larger model, we are able to analyze all items on the list [1].

Sensitivity on Recovery Rates

In order to see the advantages of the Second Model, consider the effect of changing the recovery rate for material A going through center 1. The recovery rate for this process is initially 90%. It is rather obvious that increasing the recovery rate will improve the profit if nothing else is changed, but what are the effects quantitatively?

Consider the definition equation

$$x_2 = .90x_1$$

and its corresponding constraint

$$x_2 - .90x_1 = 0$$

The dual solution corresponding to this constraint is

$$+ 6.11111$$

This is interpreted to mean that if the constraint were written

$$x_2 - .90x_1 = b_1$$

the instantaneous rate of change in profit is 6.11111 for a unit increase in b_1.

Since the recovery rate is the coefficient of x_1, we are really trying to do a sensitivity study involving a non-linearity but good approximations can be made. Since

$$x_2 = .9x_1 + b_1$$

it can be seen that increasing b_1 makes it possible to get the same output x_2 with less input x_1, i.e. an increase in recovery rate; b_1 is measured in gallons and the rate of increase in profit is in dollars per gallon.

If b_1 is increased one gallon, it will result in x_1 being reduced by $b_1/.9$ or 1.1111 b_1, providing x_2 and the recovery rate remain fixed. This can be seen since

$$x_2 = .9(x_1 - \Delta) + b_1 = .9x_1$$

$$.9x_1 - .9\Delta + b_1 = .9x_1$$

$$\text{Change in } x_1 = \Delta = \frac{b_1}{.9} = 1.111\ b_1$$

Therefore the savings will be approximately

$$1.1111\ \Delta = 1.1111\ (5 + \frac{150}{300}) = 1.111(5.5) = 6.1111$$

where the $5 is the per unit savings in raw material input and 150/300 is per unit savings in processing costs, which checks with the dual solution.

In order to translate this to a change of 1% in the recovery rate, a good approximation for the effects, for small values at least, is given by changing b_1 by an amount equal to 1% of x_1, since

$$.91x_1 = .9x_1 + b_1$$

leads to having

$$b_1 = .01x_1$$

Using the constraint

$$x_2 = .9x_1 + b_1$$

and allowing b_1 to change by an amount equal to $.01x_1$ is not equivalent to using the constraint

$$x_2 = .91x_1$$

To illustrate:

Our solution for our problem yields $\begin{cases} x_1 = 2924.0 \\ x_2 = 2631.6 \end{cases}$

Whether we use a change in b_1 or a change in recovery rate of 90%, we produce 2631.6 as the output of center 1. A reduction of 1% in input of 29.240 substituted for b_1 yields,

$$2631.6 - .9x_1 - 29.240 = 0$$

or

$$x_1 = 2891.51$$

processed at 90% recovery rate.

If we use the constraint

$$x_2 - .91x_1 = 0$$

then

$$x_1 = \frac{2631.6}{.91} = 2891.87$$

a difference of processing of .36 gallons @$6.11 or approximately $2.19.

Thus if we use changes in b_1 at 1% of x_1 and use the dual value for the constraint, the expected improvement in profit is

$$(.01)\ (2924)\ (6.1111) = 178.69$$

The new profit would then be

$$18{,}339.59 + 178.69 = 18{,}518.28$$

A solution of the problem at 91% recovery rate yields a profit of

$18,516.32

The error is small but importantly the changes in profit are in the right direction and the managerial implications of a change in recovery rate are of the correct orders of magnitude.

In a similar fashion all other recovery rate changes can be assessed by using the dual solutions. It is only necessary to change the input to any center by 1%, then multiply by its corresponding dual value to get an estimate of the effect of a change of 1% in absolute value of any recovery rate.

If one wished to find a more exact effect of the change of recovery rate, it must be done by finding the effect of changing the proper coefficient in a constraint. For the example we have studied above this amounts to changing a coefficient for the variable x_1, which is a basic variable. The model presents the necessary information for this study, but it is much more detailed.

Sensitivity on Processing Costs

In order to study the effects of processing rates, we examine the dual solutions corresponding to constraints 9 through 16. For example, if the rate of processing for center 1 were increased from 300 to 301 gallons per hour, the dual solution indicates an improvement of 50¢ in profit.

For any of the above cases it is necessary to study the ranging of the right hand side to make sure that there are no problems associated with degeneracy.

Another part of the analysis would assess the effects of changing processing costs for each of the centers. This can be done directly by a study of the ranging of the objective function for variables

$$x_{13} \text{ through } x_{20}$$

The effects of changes in raw material costs are given by looking at variables

$$x_1 \text{ and } x_9$$

and the effects of changes in selling price are given by looking at variables

$$x_7,\ x_8 \text{ and } x_{12}.$$

It should be evident that the more detailed model provides much more managerial information.

Effects on the Computer

The smaller model resulted in three variables and seven constraints and for the Computer Program used [2] at Northwestern University's Computer Center required .01 seconds of computing time. The larger model resulted in 44 variables and 24 constraints and required .1940 seconds of computing time.

The program used had capabilities of sensitivity analysis. When use was made of Ranging both the Right Hand Side and the Coefficients of the Objective Function the total computing times were .652 seconds and 2.673 seconds respectively.

Memory requirements were obviously greater for the larger model but were no threat to the solution of the problem.

The increase in both storage requirements and computing times were not insignificant but it is clear they do not impose difficulties which can not be overcome. Should the modelling of a problem along the lines illustrated in this paper lead to excessive demands on either memory or computing time, the problem can be partitioned so as to relieve the excessive demands without diminishing the advantages discussed in the paper, although more than one computer run might have to be made.

In this example satisfactory reductions can be made by eliminating the definitions and constraints in terms of the remaining variables. This reduces the problem to one of 28 variables and 16 constraints and allows one to make the sensitivity analysis on the recovery parameters.

One can by similar means eliminate other portions of the problem and thus concentrate on one or two sections at a time.

The summary computer output for each of the problems discussed in the paper are given in the Appendix.

Conclusions

The basic conclusion that can be reached is that substantial gains can be made in facility for interpretation by proper modelling of the problem. These gains are made at a cost of increased memory requirements for the computer and increased computing time.

Although consideration has been given to establishing rules for the way in which the problem should be modelled for such efficiency, no complete set has yet been established. It remains somewhat of an art or perhaps a great amount of foresight. In the examples used, an increase in both the number of variables and the number of constraints led to these efficiencies in interpretation but such is not always the case. There are other cases where the gains are made by reducing both the number of variables and number of constraints.

Complete computer outputs are not included because of the space required. These are available however.

Appendix

Small Problem

Summary of Results

Var No	Var Name	Row No	Status	Activity Level	Opportunity Cost
1	x_1	--	B	2923.9766082	--
2	x_2	--	NB	--	.5516015
3	x_3	--	B	1307.1895425	--
4	Slack	1	B	5458.5483316	--
5	Slack	2	B	3022.3684211	--
6	Slack	3	B	160423.5294118	--
7	Slack	4	B	7000.0000000	--
8	Slack	5	NB	--	2.4060028
9	Slack	6	B	700.0000000	--
10	Slack	7	NB	--	5.6997549

Maximum Value of the
Objective Function = 18339.591933

Large Problem

Summary of Results

Var No	Var Name	Row No	Status	Activity Level	Opportunity Cost
1	x_1	--	B	2923.9766082	--
2	x_2	--	B	2631.5789474	--
3	x_3	--	B	2500.0000000	--
4	x_4	--	B	2125.0000000	--
5	x_5	--	B	2125.0000000	--
6	x_6	--	B	0.0000000	--
7	x_7	--	B	1700.0000000	--
8	x_8	--	B	0.0000000	--
9	x_9	--	B	1307.1895425	--
10	x_{10}	--	B	1176.4705882	--
11	x_{11}	--	B	1000.0000000	--
12	x_{12}	--	B	800.0000000	--
13	x_{13}	--	B	9.7465887	--
14	x_{14}	--	B	5.8479532	--
15	x_{15}	--	B	10.0000000	--
16	x_{16}	--	B	5.3125000	--
17	x_{17}	--	NB	--	265.6492411
18	x_{18}	--	B	2.6143791	--
19	x_{19}	--	B	2.4509804	--
20	x_{20}	--	B	2.5000000	--
21	Artif	1	NB	--	6.1111111
22	Artif	2	NB	--	6.9005848
23	Artif	3	NB		8.9653939
24	Artif	4	NB	--	11.8942423
25	Artif	5	NB	--	11.8942423
26	Artif	6	NB	--	7.3333333
27	Artif	7	NB	--	9.2401961
28	Artif	8	NB	--	12.3002451
29	Artif	9	NB	--	.5000000
30	Artif	10	NB	--	.4444444
31	Artif	11	NB	--	.7200000
32	Artif	12	NB	--	.5500000
33	Artif	13	NB	--	-.0447121
34	Artif	14	NB	--	.6000000
35	Artif	15	NB	--	.5208333
36	Artif	16	NB	--	.6000000
37	Artif	17	NB	--	-8.9653939
38	Slack	18	B	3.6390322	--
39	Slack	19	B	.8395468	--
40	Slack	20	B	9.5490196	--
41	Slack	21	B	3.5000000	--
42	Slack	22	NB	--	2.4060028
43	Slack	23	B	700.0000000	--
44	Slack	24	NB	--	5.6997549

Maximum Value of Objective Function = 18339.591933

References

1. Hadley, *Linear Programming*, Addison-Wesley, 1962.
2. Cohen, Stein, *Multi Purpose Optimization System*, Vogelback Computing Center, Northwestern University, 1975.

INTERACTIVE COMPUTER CODES FOR MATHEMATICAL PROGRAMMING EDUCATION

Robert P. Davis and James W. Chrissis
Department of Industrial Engineering and Operations Research
Virginia Polytechnic Institute and State University
Blacksburg, Virginia

ABSTRACT

This paper describes the role of interactive computer programs in mathematical programming education. Three different categories of interactive programs are described and appropriate environments are suggested in which each should yield the greatest utility in allowing a student to enhance his computational experience with, and conceptual understanding of, an algorithm. An example execution for a program representative of each of these categories is given for illustrative purposes.

INTRODUCTION

There are essentially three problem solving methods employed in gaining computational experience with mathematical programming algorithms:

(1) Manual computation.

(2) Execution of existing "canned" programs (e.g., batch-oriented routines).

(3) Execution of interactive codes for existing algorithms (e.g., data input → decision input → solution output).

Either manual computation or execution of existing batch-oriented routines is the method most frequently employed.

Manual computation is useful in providing application experience to small problems, but even in such cases can be quite time consuming. In addition, it should be noted that minor mathematical errors may occur yielding erroneous results even though the computational methodology is correct. For this reason manual computation frequently degenerates to mere number manipulation with an abandonment of all attempts at conceptual interpretation. At the other end of the spectrum, the execution of batch-oriented routines can result in little or no understanding of the structure of an algorithm being employed to obtain a solution. The method favored by the authors is one of developing and implementing interactive codes for existing algorithms. This approach is, admittedly, an attempt at reaching a compromise between manual methods and batch-oriented routines. Interactive computing can be a very effective medium for reinforcing the learning process in mathematical programming education. Further, its implementation can significantly reduce the time required for the student to gain computational experience with existing algorithms.

DISCUSSION

Interactive computer algorithms can be separated into three broad categories:

(1) Initial data entry.
(2) Intermediate interaction (limited decision inputs).
(3) Intimate interaction in the algorithmic process.

Interactive programs requiring only initial input data can be quite useful in a problem solving environment where time is a factor and the student is already familiar with the algorithm's operation. The major utility from such programs lies in their accessibility and capacity to provide results which can be used for interpretation of model characteristics and relevance. However, such algorithms are not useful in reinforcing the solution methodology they employ. One example of such an algorithm is an interactive routine for solving linear programming problems. With this algorithm, the student enters the necessary problem data (effectiveness coefficients, input-output matrix and requirements vector) and the program returns a solution. The solution can then be used to examine model relevance or can provide a beginning for conducting postoptimality analysis. Example 1 illustrates an execution sequence for such an algorithm (2).

Algorithms which employ limited interaction during the solution process are typically those which require the student to enter: values for decisions variables, request or abort sensitivity analyses, or indicate whether optimality has been achieved. As an illustration of such algorithms, Example 2 shows an execution sequence for a dynamic programming algorithm. This algorithm is for serial systems with linear returns function and linear state transitions in a single variable at each stage. Again, such algorithms do not fully reinforce solution methodology, but they can support an awareness of an algorithm's decision process as well as illustrate inherent characteristics of the models to which they apply.

Finally, there are those programs which require an intimate interaction by the student in the algorithm's process. With such routines the student must provide the decision information necessary for the algorithm to progress. In this way, the algorithm's structure is emphasized and a more complete awareness of its operation is required. Further, an opportunity for reinforcing a conceptual understanding and interpretation of the algorithm are provided. To illustrate such programs, Example 3 shows an execution sequence for an interactive linear programming algorithm which not only requires that initial problem data be given, but further that each pivoting step in the interactive process be defined, and optimality and feasibility be identified.

With such interactive programs the major burden of mathematical manipulation is absorbed by the code. The student is required to make key decisions throughout the algorithm and is provided time to examine the consequences of these decisions. It should be noted that the student is given an opportunity to observe the advantages of machine computation in algorithmic operation; and further, interactive codes are easily structured to allow the student to observe, in the code itself, the computational building blocks which comprise the algorithm he employs.

Interactive programs of this last category can be further extended to provide error detection mechanisms. These error detections take one of two forms. First, there are those which indicate an incorrect decision but do not identify what should have been the correct response. The other not only indicates an error but also provides the correct response (for example, in an interactive LP code, a check could be made on whether the minimum non-negative theta value was selected in identifying the pivot row). It is the opinion of the authors that error checking mechanisms of this latter category are inappropriate since they eliminate an important aspect of the learning experience--that of identifying and correcting erroneous decisions made in applying a solution methodology. In fact, we question the use of any form of error detection since they do not permit the student to observe the consequences of an erroneous decision (with the LP example suggested above, a non-positive element in the solution basis).

This last point brings us to an important question which comes to the mind of the instructor who seeks to implement interactive computing in a teaching/learning activity--"What is the appropriate level (category) of interaction for the activity at hand?". For this, we have no absolute response. In general, we can state that the appropriate category of interaction is a function of the student's familiarity with an algorithm and the purpose of applying the program. If the student is in the process of learning the structure of an algorithm, the last of these categories is the most appropriate. If he is familiar with the methodology and seeks a quick solution to a particular model to examine its relevance or desires to initialize a solution to conduct sensitivity analysis, then the first category should suffice. It remains for a comprehensive investigation to provide definitive guidance to the instructor in assisting him to identify an appropriate code for his specific activity. What must be recognized is that there exists more than one level of interaction to which such programs can be brought and that an appropriate level for one activity may not be appropriate for another.

CONCLUSION

Interactive computer codes can be employed with utility to instruction in mathematical programming education. These codes can be divided into three basic categories with each representing a different level (or intimacy) of interaction. The appropriate category for a particular teaching/learning activity is not definitively established but is postulated to be a function of the student's background and the purpose for which the algorithm is being employed. As the purpose for applying the algorithm tends to be one of reinforcing the student's comprehension of a solution methodology and conceptual interpretation, the intimacy of interaction manifested in the code should increase.

REFERENCES

1. Davis, R. P. and L. D. Chapman, "An Interactive UBASIC Code for Rosen's Graident Projection Method", *CoED Transactions*, Vol, VII, No. 6, (1975).

2. Gillett, Billy E., *Introduction to Operations Research: A Computer-Oriented Algorithmic Approach*, McGraw-Hill, New York, (1976).

3. Levien, R. E. (ed.), *The Emerging Technology*, McGraw-Hill, New York, (1972).

4. Wilde, D. J. and C. S. Beightler, *Foundations of Optimization*, Prentice-Hall, Englewood Cliffs, New Jersey, (1967).

EXAMPLE 1

```
EXAMPLE FROM SECTION 3.8

   THE ORIGINAL COEFFICIENTS OF THE CONSTRAINTS

            CODE 0 ==>  <OR= CONSTRAINT
            CODE 1 ==>  >OR= CONSTRAINT
            CODE 2 ==>  = CONSTRAINT

I CODE CONSTANT  A(I,1)  A(I,2)  A(I,3)  A(I,4)  A(I,5)  A(I,6)  A(I,7)  A(I,8)

1   0    30.00    2.00    3.00
2   0    24.00    3.00    2.00
3   1     3.00    1.00    1.00

   THE COEFFICIENTS IN THE ORIGINAL OBJECTIVE FUNCTION TO BE MINIMIZED ARE:

               -6.00   -4.00


   BASIC SOLUTION   1

     XB( 1)= X( 4)=        30.00
     XB( 2)= X( 5)=        24.00
     XB( 3)= X( 6)=         3.00

     CURRENT VALUE OF THE OBJECTIVE FUNCTION IS   -0.30000000E+04

   BASIC SOLUTION   2

     XB( 1)= X( 4)=        24.00
     XB( 2)= X( 5)=        15.00
     XB( 3)= X( 1)=         3.00

     CURRENT VALUE OF THE OBJECTIVE FUNCTION IS    0.18000000E+02

   BASIC SOLUTION   3

     XB( 1)= X( 4)=        14.00
     XB( 2)= X( 3)=         5.00
     XB( 3)= X( 1)=         8.00

     CURRENT VALUE OF THE OBJECTIVE FUNCTION IS    0.48000000E+02

        THE LAST BASIC FEASIBLE SOLUTION IS OPTIMAL
        OPTIMAL VALUE OF THE ORIGINAL OBJECTIVE FUNCTION IS      -48.00
```

EXAMPLE 2

```
TYPICAL STAGE DESCRIPTION:

                         Stage Decision
                      d(I)(s(I)) = K(I)*s(I)
                          0<=K(I)<=1
                         ...............
                         :             :
      Input State        :             :    Output State
--------- s(I) ------>:               :--s(I-1)=A*s(I)+B*d(I)(s(I)) -->
                         :             :
                         ...............
                         Stage  Returns
                    r(I)=C*s(I)+D*d(I)(s(I))
                                                  Cumulative Returns
                                                f(I)=r(I)+f(I-1)(s(I-1))

If the problem is to Maximize enter a "1" for L; otherwise, enter a "0" to Minimize.
L
1
Enter the state transformation parameters: A,B as requested.
A
.7
B
.4
Enter the returns function parameters: C,D as requested.
C
.5
D
-.2
Enter the number of stages(N), up to a maximum of 6.
N
4

Stages to go:   1

Enter a value for K( 1) which will maximize:
f( 1)=  0.5000*s( 1) + ( -0.2000)*K( 1)*s( 1)
K(1)
0


Stages to go:   2

Enter a value for K( 2) which will maximize:
f( 2)=  0.8500*s( 2) + (  0.0000)*K( 2)*s( 2)
K(2)
0


Stages to go:   3

Enter a value for K( 3) which will maximize:
f( 3)=  1.0950*s( 3) + (  0.1400)*K( 3)*s( 3)
K(3)
1


Stages to go:   4

Enter a value for K( 4) which will maximize:
f( 4)=  1.3645*s( 4) + (  0.2940)*K( 4)*s( 4)
K(4)
1

DECISION SUMMARY
----------------

Stages to go     Decision
       1           0.00
       2           0.00
       3           1.00
       4           1.00

----------------
                 TOTAL RETURN =   1.6585 s( 4)
```

EXAMPLE 2 (cont.)

```
Enter the value of s( 4) as requested.
s(4)
3

                              SYSTEM PERFORMANCE

                                        K= 1
                                        .....
                                        :   :
                                   1.00 :   :     1.10
                              --------->: 4 :--------->
                                        :   :
                                        .....
                                     r=      0.30
                                                       K= 1
                                                       .....
                                                       :   :
                                                  1.10 :   :     1.21
                                             --------->: 3 :--------->
                                                       :   :
                                                       .....
                                                    r=      0.33
                                                                      K= 0
                                                                      .....
                                                                      :   :
                                                                 1.21 :   :     0.85
                                                            --------->: 2 :--------->
                                                                      :   :
                                                                      .....
                                                                   r=      0.60
                                                                                     K= 0
                                                                                     .....
                                                                                     :   :
                                                                                0.85 :   :     0.59
                                                                           --------->: 1 :--------->
                                                                                     :   :
                                                                                     .....
                         TOTAL RETURN =      1.66                                 r=      0.42
                         ____________  __________

This terminates the procedure.

**   111. XEQ "STOP".
```

EXAMPLE 3

```
DO YOU KNOW HOW TO USE THIS PROGRAM?
? N
THIS IS AN INTERACTIVE LINEAR PROGRAMMING ROUTINE
YOU MUST HAVE YOUR PROBLEM FORMULATED AS A MAXIMIZATION
PROBLEM.
THIS PROGRAM WILL ACCEPT >,< OR = CONSTRAINTS.
THE TYPE OF CONSTRAINT MUST BE INDICATED TO THE PROGRAM.
THE INDICATORS ARE: L FOR <=, G FOR >=, AND E FOR =.
ALL >= CONSTRAINTS ARE CONVERTED TO <= CONSTRAINTS.
ARE YOU READY TO USE THIS PROGRAM TO OBTAIN A SOLUTION?
? Y
ENTER THE NUMBER OF CONSTRAINTS FOLLOWED BY THE
NUMBER OF STRUCTURAL VARIABLES. SEPARATE THE NUMBERS
WITH A COMMA (,).
? 3,4
ENTER THE COEFFICIENTS OF THE OBJECTIVE FUNCTION
C(1),......C(N)
? 3,4,5,1

ENTER THE INDICATOR FOR THE TYPE OF CONSTRAINT,
 L FOR <=, G FOR >=, E FOR =, ON ONE LINE
FOLLOWED BY THE LEFT-HAND SIDE CONSTRAINT COEFFICIENTS
ONE ROW AT A TIME:A(1,1), ... ,A(1,N), ONE PER LINE.
? L
? 2,1,3,1

? L
? 1,2,4,2

? L
? 3,2,1,1

ENTER THE VECTOR OF RIGHT-HAND SIDE COEFFICIENTS
B(1),......,B(M).
? 18,26,30

CURRENT BASIC SOLUTION:
ROW              VARIABLE         VALUE
 1                 5               18
 2                 6               26
 3                 7               30
OBJECTIVE FUNCTION = 0
IS THIS SOLUTION FEASIBLE?
? Y
NON-BASIC        Z(J)-C(J)
VARIABLE           VALUE
 1                -3
 2                -4
 3                -5
 4                -1
```

EXAMPLE 3 (cont.)

```
IS THE SOLUTION OPTIMAL ?
? N
ENTER THE NUMBER OF THE NON-BASIC VARIABLE
YOU WANT TO BRING INTO THE SOLUTION.
? 3
BASIC              PIVOT              R.H.S.          THETA
VARIABLE           COLUMN             VALUE           VALUE
 5                  3                  18              6
 6                  4                  26              6.5
 7                  1                  30              30
ENTER THE NUMBER OF THE BASIC VARIABLE
TO LEAVE THE SOLUTION.
? 5
NON-BASIC          Z(J)-C(J)
VARIABLE             VALUE
 1                  .333333
 2                 -2.33333
 5                  1.66667
 4                  .666667
IS THE SOLUTION OPTIMAL ?
? N
ENTER THE NUMBER OF THE NON-BASIC VARIABLE
YOU WANT TO BRING INTO THE SOLUTION.
? 2
BASIC              PIVOT              R.H.S.          THETA
VARIABLE           COLUMN             VALUE           VALUE
 3                  .333333            6               18.
 6                  .666667            2.00002         3.00002
 7                  1.66667            24              14.4
ENTER THE NUMBER OF THE BASIC VARIABLE
TO LEAVE THE SOLUTION.
? 6
NON-BASIC          Z(J)-C(J)
VARIABLE             VALUE
 1                 -5.5
 6                  3.5
 5                 -3
 4                  2.99999
IS THE SOLUTION OPTIMAL ?
? N
ENTER THE NUMBER OF THE NON-BASIC VARIABLE
YOU WANT TO BRING INTO THE SOLUTION.
? 1
BASIC              PIVOT              R.H.S.          THETA
VARIABLE           COLUMN             VALUE           VALUE
 3                  1.5                5               3.33333
 2                 -2.5                2.99997        -1.19999
 7                  6.5                19              2.92308
```

EXAMPLE 3 (cont.)

```
ENTER THE NUMBER OF THE BASIC VARIABLE
TO LEAVE THE SOLUTION.
? 7
NON-BASIC         Z(J)-C(J)
VARIABLE            VALUE
 7                 .846154
 6                 1.38461
 5                -.461538
 4                 2.15384
IS THE SOLUTION OPTIMAL ?
? N
ENTER THE NUMBER OF THE NON-BASIC VARIABLE
YOU WANT TO BRING INTO THE SOLUTION.
? 5
BASIC             PIVOT             R.H.S.          THETA
VARIABLE          COLUMN            VALUE           VALUE
 3                 .307692           .615385         2
 2                -.846154          10.3077         -12.1818
 1                 .461538          2.92308          6.33333
ENTER THE NUMBER OF THE BASIC VARIABLE
TO LEAVE THE SOLUTION.
? 3
NON-BASIC         Z(J)-C(J)
VARIABLE            VALUE
 7                 .499999
 6                 1.5
 3                 1.5
 4                 2.5
IS THE SOLUTION OPTIMAL ?
? Y
FINAL SOLUTION VALUES AT TERMINATION
CURRENT BASIC SOLUTION:
ROW               VARIABLE          VALUE
 1                 5                 2
 2                 2                 12
 3                 1                 2.
OBJECTIVE FUNCTION = 54.
R; T=0.52/3.95 15:28:16
```

A PROBLEM SOLVING SYSTEM FOR NONLINEAR LEAST SQUARES[*]

by

Beverly A. Arnoldy
Applied Mathematics Division
Argonne National Laboratory
Argonne, Illinois 60439

and

Kenneth M. Brown[**]
Department of Computer Science
University of Minnesota
Minneapolis, Minnesota 55455

I. INTRODUCTION

In this paper, we discuss a system for solving unconstrained nonlinear least squares problems. The problem is defined as follows:

$$\min_{\underline{x}} F = \min_{\underline{x}} \sum_{i=1}^{M} [f_i(x_1,x_2,\ldots,x_N)]^2$$

where the f_i are nonlinear functions of the parameters $x_1,x_2,\ldots,x_N$. A special case of this problem, of great practical importance, is the nonlinear regression problem, where the f_i represent the residuals obtained by fitting a nonlinear model to experimental data.

The motivation behind the development of this system is to provide the user with a broad range of facilities which he can activate to ultimately enable him to obtain a solution with a minimum expenditure of computer time and his own time. We subscribe to and attempt to extend the philosophy given by Aird [1].

The four major goals of this problem solving system are:

(1) To give the user information about the behavior of his function in a region which he specifies; that is, at a set of points uniformly distributed throughout the region, to furnish the user with a sampling of F as well as with certain gradient and Hessian information -- when this information is needed and used anyway by other components of the problem solving system. One of the disadvantages of many existing optimization algorithms is that when they do not converge, the user is left with little, if any, information about the behavior of F in his region of interest.

(2) To give the user assistance from the system in choosing good starting points. Many nonlinear models are so complex that the scientist has little advance knowledge of the location of the optimum parameter values. Even in cases where the scientist has (from physical, biological, etc., considerations) a good estimate of the optimal parameters, say to within one order of magnitude, F might have a number of maxima, minima, and saddle-points close to the optimum of interest. (A saddle-point of F is a point where the gradient of F is zero, but F is neither a maximum nor a minimum.) The system provides the user with the ability to specify an entire closed region which he believes contains the minimum instead of forcing him to specify a single starting point. All too often a single "rough" starting point can produce divergence, or even convergence to a local minimum far from the minimum of interest. Once the bounds of the region have been furnished by the user as input, the system automatically narrows in on a smaller but highly promising region in which good starting points exist.

(3) To utilize a number of optimization methods so as to solve those problems for which particular optimization methods fail to find a satisfactory solution or perform poorly. For example, whereas a Levenberg-Marquardt type of method (which uses Jacobian matrix information) might be well suited for a certain scientific model throughout most of a particular region, a vastly different type of method, say a simplex method which uses no partial derivative information (only function values), might be more appropriate for some parts of that region.

(4) To systematically and automatically find the global minimum in the user's region of interest. (By "global minimum" in this paper, we mean the smallest of the local minima values in the user's region of interest.) This is accomplished in two ways. First, as described above in (2), starting guess candidates are successively narrowed down to ones most likely to yield the global minimum. Second, even after successful convergence to a minimum from one or more of the starting points, the user can pre-specify that additional promising starting points should be pursued in the attempt to find the global minimum.

---

[*] Work performed under the auspices of the U.S. Energy Research and Development Administration.

[**] A portion of the work of this author was supported by the Office of Naval Research under Grant N 00014-76-C-0329.

At present, the algorithm which implements the basic philosophy of this system is at an experimental level and should not be mistaken for the final version, nor should the FORTRAN package which realizes this implementation be construed as final code.

The paper is organized into six sections. Sections II and III describe the general algorithm, section IV discusses the current implementation of the general algorithm, section V presents a test problem with numerical results obtained from the FORTRAN package for the current implementation, and section VI describes future research plans.

II. DISCUSSION OF THE ALGORITHM

The structure of the system solver was modularized into four basic tasks or phases:

1. Pre-optimal analysis
2. Starting point generation and selection
3. Problem solution
4. Post-optimal analysis

In order to implement the concept of modularization, a control program was constructed. This program gives the user the ability to select one or more of the above phases with which to attack his problem. Each phase can be accessed alone or as part of a collection which includes other phases.

III. DESCRIPTION OF THE PHASES

Phase 1. The phase which we call "pre-optimal analysis" incorporates the ideas of (1) prescaling of the problem, and (2) verification and initialization of user-supplied routines, in particular, the verification of user-furnished exact analytic partial derivatives by means of finite-difference techniques.

Phase 2. The motivation behind phase two is derived from the situation a user faces when attacking a problem about which he has very little information. He may not be able to choose a starting point sufficiently close to the minimum. Perhaps at best he can supply upper and lower bounds on a region in which he suspects a minimum to exist. In order to handle situations such as this, a point-dispersion algorithm can be used to generate a specified number of points which are uniformly distributed in a closed rectangular region that the user has defined. Let the set $S = \{s_1, s_2, \ldots, s_{N_1}\}$ denote these points. Now evaluate F on the set S and select a subset, $T = \{t_1, t_2, \ldots, t_{N_2}\} \subset S$, with $N_2 \leq N_1$, which consists of the ordered points for which F has ascending values; i.e.

$$F(t_1) \leq F(t_2) \leq \ldots \leq F(t_{N_2}) .$$

At this stage, t_1 appears to be our most promising point for producing a minimum of F; however, as Figure 1 (for a one-dimensional problem) shows, t_1 might not lie in the valley containing the minimum of interest.

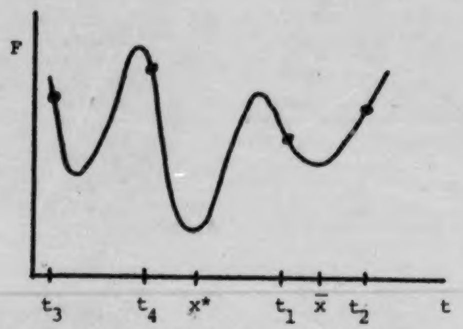


Figure 1

All too many problem solving systems would use t_1 as a starting point for an optimization procedure which would then converge to $\bar{x}$. Our approach follows that of Aird [1], namely, from each of the points $t_1, t_2, \ldots, t_{N_2}$, we use our best (fastest) optimization algorithm for a small number of iterations (we have used three for this number with excellent success), producing yet a third set of points, $P = \{p_1, p_2, \ldots, p_{N_2}\}$. Again, we evaluate F on the set P and select a subset $R = \{r_1, r_2, \ldots, r_{N_3}\} \subset P$, with $N_3 \leq N_2$, which consists of points ordered so that $F(r_1) \leq F(r_2) \leq \ldots \leq F(r_{N_3})$. The points in R can then be used successively as "final starting points". In practice, the restricted set R has yielded excellent starting points for use in phase three. In the figure above, please note that any good gradient (descent) procedure would result in $r_1 = x^*$ being used as the first (most promising) starting point in phase three.

Phase 3. The methodology of phase three concentrates on problem solution: convergence to a minimum. Two major considerations have arisen in the development of a general algorithm:

(1) What strategy can be used to blend existing complementary optimization techniques in such a way as to achieve a balance between efficiency and robustness? (By "efficiency" we mean minimization of speed of execution and core storage requirements. By "robustness" we mean the ability of a method to converge from a very wide distribution of starting points for a variety of functions.)

(2) What criteria are involved in discontinuing the use of one method and initiating the use of another?

Ideally, we would like to use one method that is both efficient and robust. However, highly efficient methods are seldom robust, and vice versa. A local optimizer is classified as an efficient method and is one which converges very rapidly for starting points that are close to (local to) the minimum. An example of a local optimizer is the Gauss-Newton method applied to finding the minimum of quadratic function (from any starting point). In contrast, a robust method will usually converge to a solution from a wide collection of starting points, although the convergence rate is generally much slower than when using a local optimizer. An example of a robust method is a nonlinear simplex algorithm.

These considerations led to the development of a hierarchical structure in which the most efficient method is at the top of the hierarchy while less efficient but more robust methods follow, and finally, the most robust method is at the bottom of the hierarchy. The structure we have adopted utilizes an efficient method as the primary method of problem solution. However, if the efficient method fails to make reasonable progress towards the minimum, alternative methods which are less efficient but more robust and which have completely different structural properties should be available. For example, it makes little sense to switch back and forth between local methods such as Gauss-Newton, Levenberg-Marquardt, and Steepest Descent if the Levenberg-Marquardt algorithm is not converging properly, since the Levenberg-Marquardt algorithm is already a compromise between the other two. Instead, one should try a method having a completely different structure, perhaps, in this case, a Quasi-Newton method or one of the better heuristic search procedures.

In our implementation the local method is used and reused first, and only when the local method fails to make progress do we switch to a more robust method.

Phase 4. In phase four, which we call "post-optimal analysis", it is verified whether or not the solution obtained in phase three is indeed the minimum, as opposed to, say, a saddle-point.

IV. CURRENT IMPLEMENTATION

The current implementation of the system solver is described below with the aid of the flow diagrams shown in Figures 2 through 5.

The control program (see Figure 2) directs the flow of control of the system. This is accomplished by two input parameters, INOPT and OUTOPT, each of which takes on a value from one to four. The value of INOPT specifies which phase the user wants to access first, while the value of OUTOPT defines the last phase he wants to access. For example, if the user requested execution of all four phases, INOPT would be set to one, and OUTOPT would be set to four. However, if he only wanted to generate starting points, both INOPT and OUTOPT would be set to the value of two. The control program also enables the user to specify more than one starting point when attempting to find the minimum. (These points will be used on successive attempts by phase three.) The ability to use more than one starting point has a twofold advantage:

(1) It increases the possibility of converging to a global minimum from at least one starting point.

(2) If the user's problem is very complex and it is difficult to find even a

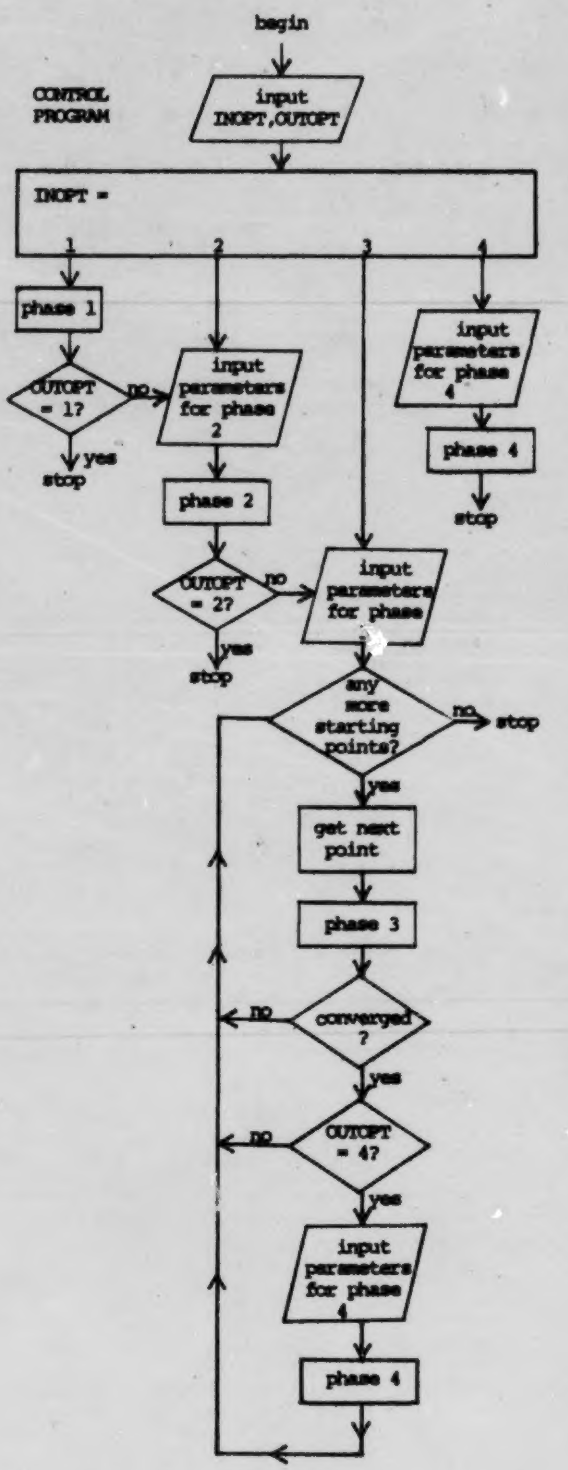

begin
CONTROL PROGRAM
input INOPT,OUTOPT
INOPT =
1
2
3
4
phase 1
OUTOPT = 1?
no
yes
stop
input parameters for phase 2
phase 2
OUTOPT = 2?
no
yes
stop
input parameters for phase 4
phase 4
stop
input parameters for phase
any more starting points?
no
stop
yes
get next point
phase 3
converged ?
no
yes
OUTOPT = 4?
no
yes
input parameters for phase 4
phase 4

Figure 2

local minimum, the possibility of converging to a local minimum increases when using more than one starting point.

At present, phase one has not been implemented.

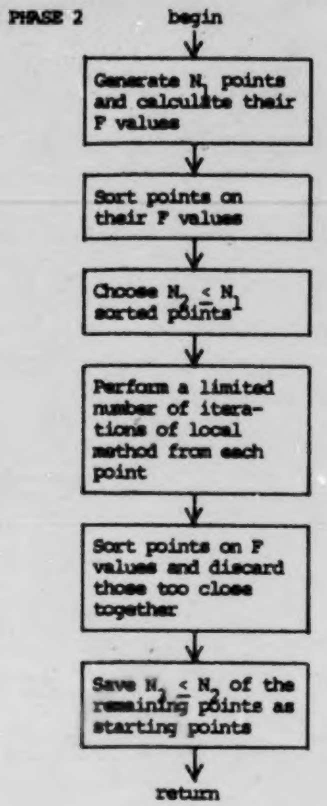

PHASE 2
begin
Generate N_1 points and calculate their F values
Sort points on their F values
Choose $N_2 \leq N_1$ sorted points
Perform a limited number of iterations of local method from each point
Sort points on F values and discard those too close together
Save $N_3 \leq N_2$ of the remaining points as starting points
return

Figure 3

The construction of phase two (see Figure 3) is somewhat more detailed than its description given above in section III, but it parallels the general algorithm. N_1, N_2, and N_3 are user-supplied parameters. The methods currently used in phase two are the Aird and Rice point-dispersion algorithm [2], and Brown's derivative-free modification of the Levenberg-Marquardt method [3].

The current implementation of phase three (see Figure 4) utilizes a local optimizer as the primary method for solution and an alternative method, a nonlinear simplex algorithm. The local method is used initially for a specified number of iterations. When the method does not converge to a minimum, the progress of the method is evaluated. If the results indicate that it is performing well enough to continue, the flow of control returns to the local optimizer. If the progress evaluation indicates that the method is performing poorly, the nonlinear simplex method is employed. If this method does not converge to a minimum in a specified number of iterations, an evaluation of its performance is necessary. The flow of control returns to the local optimizer if the results indicate a significant degree of progress. However, if the simplex method has performed unsatisfactorily, the flow of control returns to the control program with an indication that a solution could not be found. At this point, the next most promising starting point candidate is used and phase three is re-entered. Successful progress of both optimizers is dependent upon a significant decrease in the F value or in the norm of the gradient. The local method currently used is Brown's method [3]. The nonlinear simplex algorithm used is Parkinson's modified version of Nelder and Mead's nonlinear simplex algorithm [4,5].

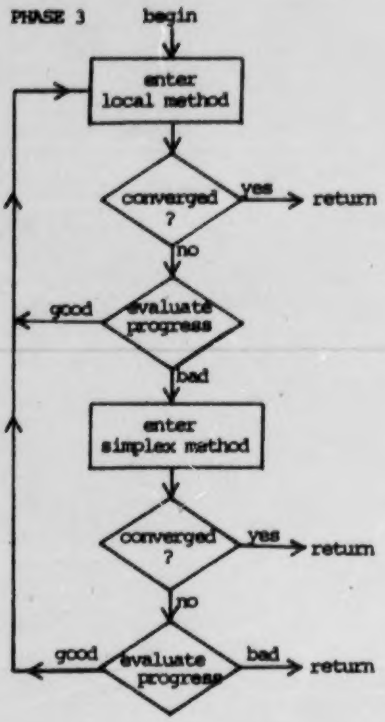

PHASE 3
begin
enter local method
converged ?
yes
return
no
good
evaluate progress
bad
enter simplex method
converged ?
yes
return
no
good
evaluate progress
bad
return

Figure 4

Post-optimal analysis in phase four (see Figure 5) is accomplished by examining points in a neighborhood of the solution obtained in phase three and then comparing the F values of these points against the F value at the solution point.

V. A TEST PROBLEM AND NUMERICAL RESULTS

A FORTRAN program based upon the current implementation of the problem solving system has been run successfully on a collection of standard test problems. (Those results will be documented elsewhere.) For the purposes of this discussion, we have constructed a new test problem whose geometry would challenge the ability of the system to find a global minimum.

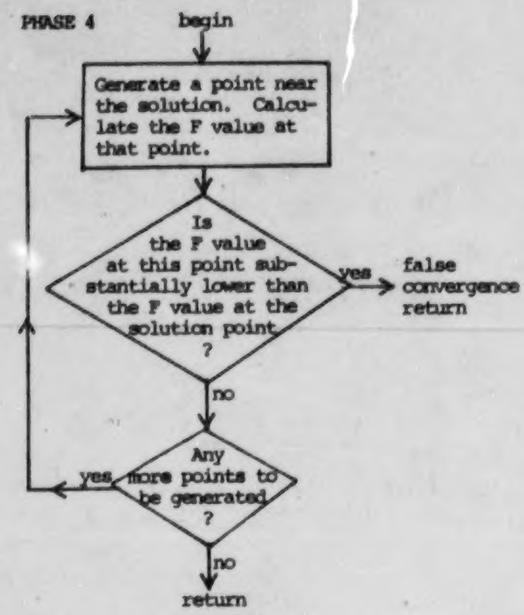

PHASE 4
begin
Generate a point near the solution. Calculate the F value at that point.
Is the F value at this point substantially lower than the F value at the solution point ?
yes
false convergence return
no
Any more points to be generated ?
yes
no
return

Figure 5

The test problem is given by:

minimize $F(x,y)$ where

$$F = .0001 * (f_1^2 + f_2^2),$$

$$f_1 = 200. - 175. * [\exp(-(x - 17.)^2) + \exp(-(y - 17.)^2)],$$

and

$$f_2 = 5. * [(x - 12.)^2 * (x - 23.) * (x - 17.) + (y - 12.)^2 * (y - 23.) * (y - 17.)].$$

Note that F is symmetric in x and y.

A three-dimensional plot of F over the range of interest is given in Figure 8. In order to better exhibit the topography of F around the global minima, a contour plot is shown in Figure 9.

The reason that F is challenging, especially for local optimization methods, is that the region, x = (11,24) and y = (11,24), contains a number of local minima and saddle-points, including a saddle-point close to the global minima. Specifically, the points given (approximately) by (12,12), (12,23), (23,12), and (23,23) are all local minima of F and at those points the value of F is 4. Similarly, the points given (approximately) by (12,17), (17,12), (17,23), and (23,17) are also local minima of F; at each of these points F takes on the value of .0625. The troublesome saddle-point occurs at (17,17) where F assumes the value of 2.25; this saddle-point is near the global minima which are given (approximately) by (17.61014224,16.110990468) and the symmetrically placed point, (16.110990468,17.61014224). At the global minima F has the value of zero.

An observation to be noted is that even finer contour plots than the one given in Figure 9 failed to expose the locations of the global minima. Graphical techniques have merit in showing the overall general behavior of a function, but, as this test function indicates, graphics cannot be relied upon to solve optimization problems.

Experiment #1. In order to obtain a measure of the robustness of the system solver vs. the robustness of the stand-alone methods of which it is composed, we ran (a) the system solver, (b) the local optimizer (Brown's method [3]), and (c) the simplex method [4,5] from first 10, then 20, and finally 40 starting points uniformly distributed in the region of interest, x = (1,31) and y = (1,31). The results are summarized in the table given in Figure 6.

$N_1 =$ $N_2 =$ N_3	SYSTEM SOLVER		LOCAL OPTIMIZER		SIMPLEX METHOD	
	SUCCESS	FAILURE	SUCCESS	FAILURE	SUCCESS	FAILURE
10	10	0	8	2	8	2
20	20	0	18	2	20	0
40	40	0	35	5	22	8

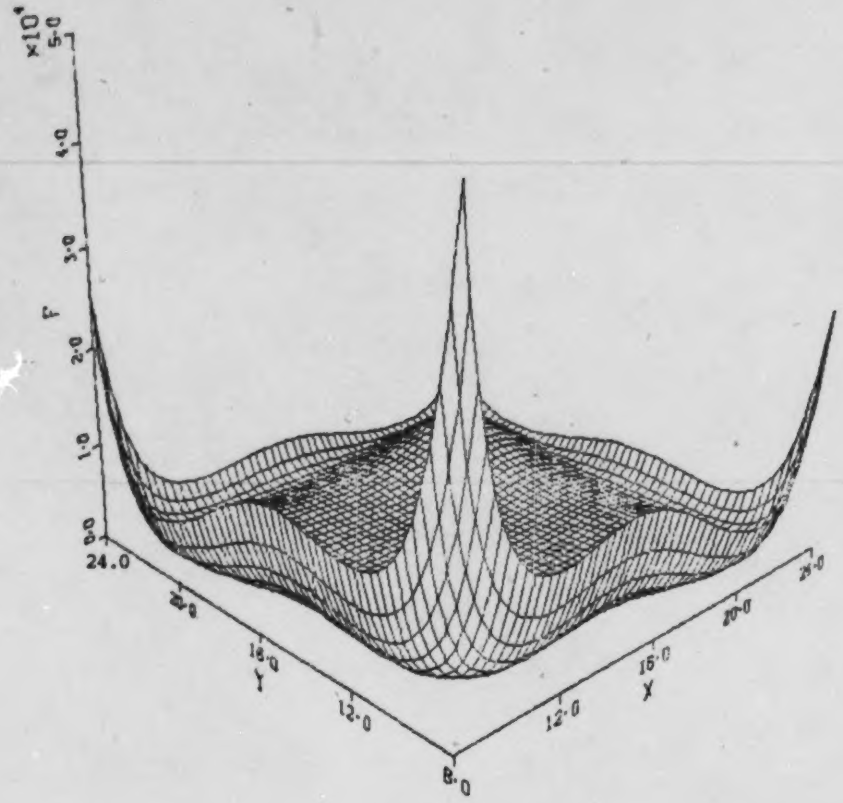
Figure 6

The word "success" in that table means that the method converged to a local minimum (which in some cases corresponded to a global minimum), whereas the word "failure" means that the method converged to a saddle-point, diverged, or failed to converge in the maximum number of function evaluations or maximum number of iterations that were allowed.

Experiment #2. The purpose of this experiment was to see how many points (N_1) had to be uniformly scattered throughout the region in order for the system solver to produce a global minimum as distinct from a local minimum. As Figure 7 indicates,

N_1	N_2	N_3	Value of F at minimum
5	5	1	.0625
10	5	1	4.
20	5	1	0.
40	5	1	0.

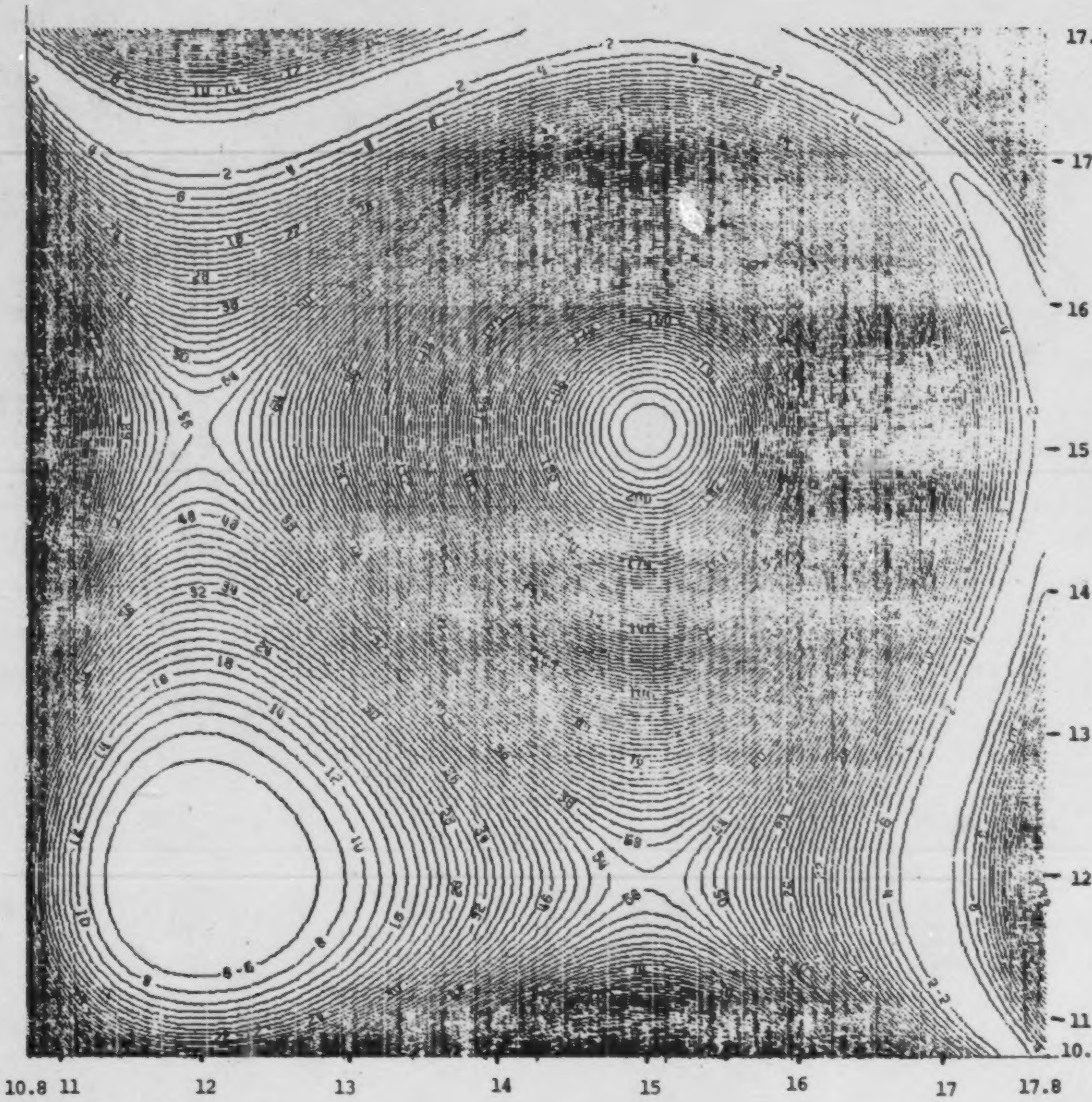
Figure 7



3-D GRAPH OF THE TEST FUNCTION

X = (8,24), Y = (8,24)

Figure 8



CONTOUR PLOT OF THE TEST FUNCTION

X = [10.8,17.8], Y = [10.8,17.8]

CONTOUR INTERVAL = 2.00 UNITS

Figure 9

as soon as 20 points (or more) are initially scattered, the system solver converges to a global minima. In order to illustrate this experiment in greater detail, let us consider the case in which $N_1 = 20$, $N_2 = 5$, and $N_3 = 1$ in Figure 7 (see also the discussion of phase two in Section III of this paper). In phase two of this algorithm, an input of $N_1 = 20$, causes 20 points to be uniformly scattered by the Aird-Rice algorithm [2] in the region of interest which is x = (1,31) and y = (1,31). The location of these points is shown in Figure 10. The numbering of the points given in Figure 10 corresponds to the order in which they were produced by the Aird-Rice Method [2]. Phase two now evaluates F at each of these 20 points. The input parameter value, $N_2 = 5$ (see Figure 9) now causes phase two to take the 5 points (of the original 20) having the smallest F value and perform 3 iterations of the local method from each of these 5 points. Once again, the F values of the 5 resulting points are computed. Finally, the input, $N_3 = 1$, (see Figure 7) causes the selection of the one point which has thus far produced the smallest F value to be used as THE starting point for phase three. That point caused phase three to converge to the global minimum which had the value of zero. It is of interest that the point (19,10) labeled "6" in Figure 10 had the smallest F value of the original $N_1 = 20$ scattered points, but if that point had been used directly as the starting point for phase three, the system solver would have converged to the local minimum of F at (12,12) with the corresponding F value of 4. On the other hand, the point (19,16) labeled "3" in Figure 10 only ranked third on the list of the 5 best points, based upon F values only; however, when that point benefited from being run through the 3 iterations of the local method, the resulting point had the smallest F value and was used in phase three, which then produced the global minimum, (17.61014224,16.110990468) with a corresponding F value of zero.

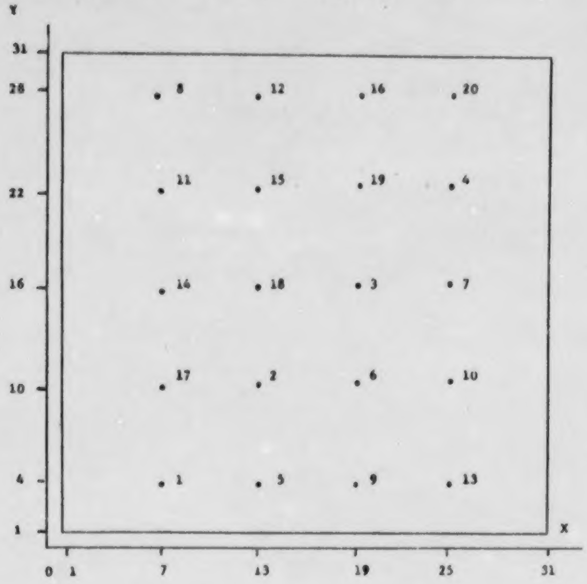


Figure 10

In summary, the system solver displayed a high success rate (see Figure 6) and, once enough points were used in the initial scattering by phase two (i.e., once N_1 was large enough), the system solver found a global minimum (see Figure 7).

VI. FUTURE RESEARCH AND PLANS FOR THE FIRST DISTRIBUTED CODE

The long range plans for the problem solving system include:

(1) The ability to solve nonlinear unconstrained optimization problems and nonlinear systems of equation problems in addition to the current ability of solving nonlinear least squares problems.

(2) The ability to implement any of a number of hierarchical structures in phase three. Initially, we shall explore a three-tiered structure in which the top level (the level which is used and reused first) includes methods which require Hessian information. The second level of methods utilize gradient information and the bottom level methods utilize only function values. Typically, the methods at the top level are the most efficient whereas the methods at the bottom level are the most robust.

(3) The ability to plug in any local method or robust method into the appropriate boxes in phase three without the user having to reprogram any of the rest of the problem solving system. Similar abilities apply to phases two and four.

(4) The development of a control language to allow the user to make requests of the system in simple meaningful command statements.

(5) The creation of a fully modularized package.

In order to achieve one goal of producing and distributing useful code by the end of 1977, we shall concentrate on a specific problem area, the unconstrained minimization of a sum of squares of nonlinear functions. We have adopted the following implementation strategy:

Phase 1. As we plan to allow derivative or derivative-free methods in phase three, we shall implement code to, at the user's option, verify the correctness of the partial derivatives which he furnishes. Again, at the user's option, the system will automatically provide default values for the required input parameters. Finally, a scaling strategy involving the diagonal of the inverse Hessian matrix of F will be tested and implemented if successful.

Phase 2. This phase will remain as it is now, utilizing the Aird-Rice point-dispersion algorithm [2] followed by three iterations of

Brown's method [3].

Phase 3. We shall adopt the three-tiered hierarchy, using Brown's Levenberg-Marquardt type of method [3] at the top of the hierarchy, the method of descent (searching in the direction of the negative of the gradient) at the second level and again Parkinson's nonlinear simplex method [4,5] at the bottom level. The flow of control which we shall implement in phase three is given in Figure 11.

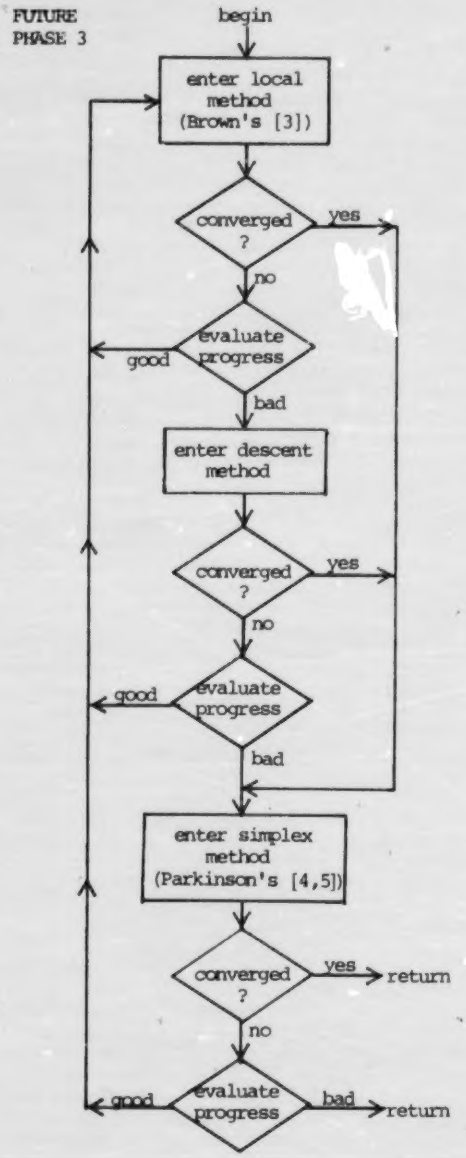

FUTURE
PHASE 3
begin
enter local method (Brown's [3])
converged ?
yes
no
evaluate progress
good
bad
enter descent method
converged ?
yes
no
evaluate progress
good
bad
enter simplex method (Parkinson's [4,5])
converged ?
yes
return
no
evaluate progress
good
bad
return

Figure 11

Note. The reason that Brown's method [3] and the method of descent transfer control to the simplex method upon successful convergence is to avoid the difficulties associated with saddle-points. Our numerical experiments have indicated that gradient and Levenberg-Marquardt type of methods will readily converge to a saddle-point; however, a few iterations of the simplex method are sufficient to move away from such a point. Obviously, there will be sufficient tests made to avoid infinite looping (upon successful convergence) in phase three.

Phase 4. We shall compare an Aird-Rice [2] scattering approach with a cyclic coordinate search in which the coordinate axes have been rotated to correspond to information obtained from the statistics of the nonlinear regression fit. The best of these approaches will be used in phase four.

ACKNOWLEDGEMENT

The authors are most grateful to Dr. J. L. Nazareth whose suggestions aided in the development of the system solver.

REFERENCES

[1] Aird, T. J., "Computer Solution of Global Nonlinear Least Squares Problems," Ph.D. Thesis, Purdue University (1973).

[2] Rice, J. R. and Aird, T. J., "Systematic Search in High Dimensional Sets," (to appear).

[3] Brown, K. M., "NLNREG: A FORTRAN Package for Nonlinear Regression," (to appear).

[4] Parkinson, J. M. and Hutchinson, D., "An Investigation into the Efficiency of Variants of the Simplex Method," Numerical Methods for Non-linear Optimization, F. A. Lootsma, Ed., Academic Press, New York, pp. 115-135 (1972).

[5] Nelder, J. A. and Mead, R., "A Simplex Method for Function Evaluation," Computer Journal, Vol. 7, pp. 308 (1965).

AN ITERATIVELY REWEIGHTED LEAST SQUARES SYSTEM

Virginia Klema*

National Bureau of Economic Research, Inc.
Computer Research Center
575 Technology Square
Cambridge, Massachusetts 02139

Abstract

The purpose of this paper is to present a description of a system of subroutines to compute solutions of the iteratively reweighted least squares problem where the weights themselves are functions of the scaled residuals. Starting points for the iterations are the ordinary least squares solution, the overdetermined solution in the L_1 norm, or starting points specified by the user.

Introduction

The iteratively reweighted least squares algorithms are a part of robust regression where "robustness" is used in the statistical sense of relative insensitivity to moderate departures from assumptions. The experimental system of subroutines that is used to compute the solution to the iteratively reweighted least squares problem is modular mathematical software written as a collection of Fortran subroutines. The subroutines are designed to operate efficiently and reliably on computing machines of the major manufacturers. The specific machines to which we refer are CDC 6600/7600, Honeywell 6000, IBM 360/370, PDP 10, and Univac 1108.

The software for solving iteratively reweighted least squares problems represents interdisciplinary research in numerical analysis, robust statistics and quality software and, as such, represents the combined work of many people. The basic design of the iteratively reweighted least squares algorithm and the computation of the default "tuning constants" for the various weight functions was done by Paul Holland. The convergence criterion for iteratively reweighted least squares was devised by John Dennis. The L_1 start from the overdetermined solution in the L_1 norm was provided by Richard Bartels. The subroutines for the software for the stem and leaf display were done by David Hoaglin and Stan Wasserman. The design of the interactive driver program was based on the advice of David Hoaglin and Roy Welsch. Technical and programming assitance was provided by David Coleman, Neil Kaden, and Sandra Moriarty. Valuable discussions continue to be held with David Gay and Richard Hill.

The substantial contributions of Gene Golub have been central to this work. His encouragement to us, his constructive work on the numerical stability of the algorithms, and his continuing exposition that crosses the boundary of robust statistics and numerical algebra constitute an essential resource.

Section 1

The method of least squares is versatile and numerically stable when computationally stable methods are used, i.e. [1,4]. Nonetheless, least squares does not give very much information about outliers or leverage points if one looks simply at the coefficients x of $b = Ax + e$. In the notation $b = Ax + e$, b is an $m\text{x}1$ vector of observations, A is an $m\text{x}n$ data or design matrix, x is an $n\text{x}1$ vector of parameters, e is an $m\text{x}1$ vector. We recognize that the usual statistical notation is $y = X\beta + e$ where y is $n\text{x}1$, X is $n\text{x}p$, B is $p\text{x}1$, and e is $n\text{x}1$. In the iteratively reweighted least squares subroutines the software for least squares model fitting technology has been extended to provide more information about the data and to provide a vehicle for handling large residual problems.

The ordinary least squares problem is

$$\min_{x} \sum_{i=1}^{m} \left(\frac{r_i(x)}{s} \right)^2 \quad \text{where}$$

r is the residuals, $b-Ax$, and s is a scale.

The weighted least squares problem is

$$\min_{x} \sum_{i=1}^{m} W_i \left(\frac{r_i(x)}{s} \right)^2$$

which is solved by using ordinary least squares with $W^{1/2}A$ and $W^{1/2}b$. W is a diagonal matrix of weights that are functions of scaled residuals.

The iteratively reweighted least squares problem assumes a start. Presently the L_2 start can be computed from MINFIT from EISPACK II followed by MINSOL which determines the best approximate rank of A and computes the least

---

*This work was supported in part by the National Science Foundation under Grant No. MCS76-11989.

squares solution. Alternatively the L_2 start can be computed from the subroutines QRF, an orthogonal decomposition based on Householder tranformations, followed by QRSOL which solves $Ax = b$ from the output of QRF. An L_1 start can be obtained from the subroutines L_1, by Richard Bartels, which compute the overdetermined solution in the L_1 norm. Given the starting solution, the scale is determined from subroutine SMAD to get the median $\frac{|r_i|}{.67449}$, $r_i \neq 0$. The scaled residuals are formed by subroutine SCIMAD. The weighting matrix $W^{1/2}$ is determined from any one of the eight subroutines that compute the weight functions. Thus, given $\hat{X}^{(0)}$ from L_2 or L_1, the problem is iterated to obtain $\hat{X}^{(k+1)} = (A^T W^{(k)} A)^I W^{(k)} b$ using MINFIT or QR factorizations.

To test convergence, after the k^{th} iteration, we compute

$$\frac{\left(\left(\left(W^{(k+1)}\right)^{1/2} A_i\right)^T \left(\left(W^{(k+1)}\right)^{1/2} R^{(k)}\right)\right)}{||\left(W^{(k+1)}\right)^{1/2} A_j||_2 \; ||\left(W^{(k+1)}\right)^{1/2} R^{(k)}||_2}$$

where $||\cdot||$ is the Euclidean norm.
Subroutine WGRAD1 computes the gradient and subroutine WGRAD2 computes a scale independent measure of the gradient.

Section 2

The term "robustness" has a common thread of meaning that carries through statistical robustness, computational stability, and reliable software. From the standpoint of reliable software, moderate departures from assumptions means that the performance of the software shall be unaffected (in the sense that performance will not be degraded) by the environment in which the software is run, the compiler from which code is generated, or the applications system in which the software is imbedded. In particular, we program to avoid abnormal system interruption or termination. Cody [2] has given an excellent exposition of reliable software and has provided more details in his position paper for the workshop on robust software, Computer Science and Statistics, Ninth Annual Symposium on the Interface.

Throughout the work on iteratively reweighted least squares heavy emphasis has been placed on modular subroutines. For example the convergence criterion can be changed, weight functions can be added, and numerical equilibration (for columns of the A matrix) can be invoked. Optionally the "hat" matrix which is the projection matrix,

$$P_A = A(A^T A)^I A^T,$$

is obtained as UU^T from the singular value decomposition or QQ^T from Householder transformations. If desired, the stem-and-leaf display of the residuals is provided. An interactive driver program is used to print the singular values, the L_2 condition number, select one or more weighting functions, display residuals, monitor convergence, and optionally select the display of the diagonal or the upper triangle of the "hat" matrix and the histogram for the stem-and-leaf.

Our approach to programming design included documentation for use, and flow of program control, as comments in the program. We use a declaration checker to be sure that all variables have been declared. The Fortran verifier, PFORT, from Bell Telephone Laboratories, was used to check the subroutines, and the Fortran converter, from IMSL, was used to generate the Fortran code for non-IBM machines.

The explicit weight functions that we have used are listed in the Appendix I, Table 1. The subroutines, with the exception of those required for the L_1 start, are listed in Table 2. Typical convergence quantities for data from [3] are listed in Table 3. Appendix II shows a sample experimental program for one of the weight functions.

References

1. Businger, P. and Golub, G. H., "Linear Least Squares Solutions by Householder Tranformations," in Linear Algebra, Wilkinson, J. H., and Reinsch, C., Springer-Verlag, 1971.

2. Cody, W. J. [1974], "The Construction of Numerical Subroutine Libraries," SIAM Review 16, 36-46.

3. Draper, N. R. and Smith, H., Applied Regression Analysis, John Wiley & Sons, 1966, p.352.

4. Golub, G. H., and Reinsch, C., "Singular Value Decomposition and Least Squares Solutions," in Linear Algebra, Wilkinson, J. H., and Reinsch, C., Springer-Verlag, 1971.

Appendix I

Table 1

Examples of weight functions (where u = scaled residual) and the default tuning constant for each weight function.

ANDREWS $$w_A(u) = \begin{cases} \dfrac{\sin\left(\frac{u}{A}\right)}{\left(\frac{u}{A}\right)} & |u \leq \pi A \\ 0 & |u > \pi A \end{cases}$$

$A = 1.339$

BIWEIGHT $$w_B(u) = \begin{cases} \left[1 - \frac{u}{B}^2\right]^2 & |u \leq B \\ 0 & |u > B \end{cases}$$

$B = 4.685$

CAUCHY $w_C(u) = \dfrac{1}{1+\left(\frac{u}{C}\right)^2}$

$C = 2.385$

FAIR $w_F(u) = \dfrac{1}{1+\left|\frac{v}{F}\right|}$

$F = 1.400$

HUBER $w_H(u) = \begin{cases} 1 & |u| \leq H \\ \frac{H}{|u|} & |u| > H \end{cases}$

$H = 1.345$

LOGISTIC $w_L(u) = \dfrac{\tanh\left(\frac{u}{L}\right)}{\left(\frac{u}{L}\right)}$

$L = 1.205$

TALWAR $w_T(u) = \begin{cases} 1 & |u| \leq T \\ 0 & |u| > T \end{cases}$

$T = 2.795$

WELSCH $w_R(u) = e^{-\left(\frac{u}{R}\right)^2}$

$R = 2.985$

NAME	DESCRIPTION
****	***********
EQ01	MODIFIED ROW-INF-EQUILIBRATION
EQ02	COLUMN (MAX. ELEMENT) EQUILIBRATION
EQ03	ROW (MAX. ELEMENT) EQUILIBRATION
EQ04	COLUMN (SQRT. SUM OF SQUARES) EQUILIBRATION
EQ05	ROW (SQRT. SUM OF SQUARES) EQUILIBRATION
EUNORM	EUCLIDIAN (SQRT. SUM OF SQUARES) NORM
HMAT	FORMS DIAGONAL OF H-MATRIX (U*U-TRANS)
HMATQR	FORMS DIAGONAL OF H-MATRIX (Q*Q-TRANS)
ISORT1	SHELL SORT (DECREASING) USING INDIRECTION
ISORT2	SHELL SORT (INCREASING) USING INDIRECTION
MINFIT	SINGULAR VALUE DECOMPOSITION A=U*SIGMA*V-TRANSP
MINSOL	SOLVES AX=B GIVEN OUTPUT FROM MINFIT
QRF	QR DECOMPOSITION, Q ORTHOGONAL TRANSFORMATIONS
QRSOL	SOLVES AX=B USING QRF
RESIDL	COMPUTES REDISUAL B-AX
SCLMAD	SCALE RESIDUALS BY SCALING FACTOR
SLDSPY	DOES STEM AND LEAF DISPLAY (CALLS OTHERS)
SILEAF	DETERMINES STEMS AND LEAVES
SLPRNT	PRINTS STEM AND LEAF DISPLAY
SLSCAL	DETERMINES SCALE FACTOR AND UNIT FOR DISPLAY
SLSCRT	SHELL SORT IN INCREASING ORDER
SMAD	DETERMINES MAD SCALING FACTOR
WANDRW	ANDREWS WEIGHTING FUNCTION
WBIWGT	BIWEIGHT (BISQUARE) WEIGHTING FUNCTION
WCAUCH	CAUCHY WEIGHTING FUNCTION
WELSCH	WELSCH WEIGHTING FUNCTION
WFAIR	FAIR WEIGHTING FUNCTION
WGRAD1	COMPUTES GRADIENT
WGRAD2	COMPUTES SCALE INDEPENDANT MEASURE OF GRADIENT
WHUBER	HUBER WEIGHTING FUNCTION
WLOGIS	LOGISTIC WEIGHTING FUNCTION
WTALWR	TALWAR (ZERO-ONE) WEIGHTING FUNCTION

Table 3

The data from [3], suggested by Paul Holland as test data is a 25 x 10 data matrix A with $A_{i1} = 1.0$. The L_2 condition number is

$$\frac{\sigma_{max}}{\sigma_{min}} = \frac{.396 \times 10^3}{.789 \times 10^{-1}} .$$

The maximum diagonal element of the H, "hat" matrix is .85. Based on functions of the scaled residuals, the effect of the weight functions is to down weight some of the observations.

For the weight functions listed below the maximum element in magnitude of the scale-free measure of the gradient after iteration 1 and after iteration 10 is as follows.

		after iter 1	after iter 10
Andrews	L_1 start	.383	$.244 \times 10^{-7}$
	L_2 start	.104	$.177 \times 10^{-4}$
Biweight	L_1 start	.383	$.245 \times 10^{-7}$
	L_2 start	.105	$.184 \times 10^{-4}$
Huber	L_1 start	.383	$.609 \times 10^{-8}$
	L_2 start	$.894 \times 10^{-1}$	$.435 \times 10^{-5}$

```
FILE: BIWGT    FORTRAN  *              CORNELL VM/370 3.4

      SUBROUTINE WBIWGT(N,U,CONST,SQW)
C     *****PARAMETERS:
      INTEGER N
      REAL*8  U(N),CONST,SQW(N)
C     *****LOCAL VARIABLES:
      INTEGER I
      REAL*8  DELIM,UBETA,UI,PROD
C     *****FUNCTIONS:
      REAL*8  DABS
C
C
C     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
C
C
C     *****PURPOSE:
C     THIS SUBROUTINE PRODUCES THE SQUARE ROOTS OF THE WEIGHTS
C     DETERMINED BY THE INPUT VECTOR U OF PREVIOUSLY COMPUTED
C     SCALED RESIDUALS AND THE BIWEIGHT (BISQUARE) WEIGHT FUNCTION.(1)
C
C     *****PARAMETER DESCRIPTION:
C     ON INPUT:
C
C        N MUST BE SET TO THE NUMBER OF ELEMENTS IN THE VECTORS U AND
C             SQW;
C
C        U CONTAINS THE STANDARDIZED RESIDUALS FROM A PREVIOUS LINEAR
C             FIT.  THAT IS, U(I) = R(I) / S WHERE R(I) IS THE I-TH
C             RESIDUAL FROM A LINEAR FIT, R(I) = Y(I) - YFITTED(I),
C             AND S = S(R) IS A RESIDUAL SCALING FUNCTION (E.G. S COULD
C             BE THE OUTPUT OF THE FORTRAN SUBROUTINE SMAD).
C
C        CONST IS THE 'TUNING CONSTANT' FOR THE WEIGHT FUNCTION
C             W(U).  (SEE APPLICATION AND USAGE RESTRICTIONS)
C
C
C     ON OUTPUT:
C
C
C
C        SQW CONTAINS A VECTOR OF THE SQUARE ROOTS OF THE WEIGHTS
C             DETERMINED BY THE SCALED RESIDUALS AND THE WEIGHTING
C             FUNCTION.
C
C     *****APPLICATION AND USAGE RESTRICTIONS:
C     THE ROOT-WEIGHTS ARE NEEDED FOR THE COMPUTATION OF THE
C     ITERATIVELY REWEIGHTED LEAST SQUARES ESTIMATES USING THE
C     FORTRAN SUBROUTINES MINFIT AND MINSOL.  IN THIS COMPUTATION
C     SQW(I) MULTIPLIES THE CORRESPONDING ROWS OF THE X-MATRIX
C     AND THE Y-VECTOR. (1)
C
C     THE LARGER THE VALUE OF CONST, THE MORE NEARLY ALL THE VALUES
C     OF W(U) WILL EQUAL UNITY.
C
C     IF CONST IS TAKEN TO BE VERY SMALL IT IS POSSIBLE TO PRODUCE A
```

```
C       VECTOR OF ROOT-WEIGHTS ALL OF WHICH EQUAL OR NEARLY EQUAL
C       ZERO, AND THIS WILL BE USELESS AS INPUT TO THE WEIGHTED LEAST
C       SQUARES COMPUTATIONS.
C
C       IF A TUNING CONSTANT VALUE OF 4.685 IS USED, UNDER THE
C       ASSUMPTION OF GAUSSIAN ERRORS, THE RESULTING ESTIMATOR
C       WILL HAVE 95 PERCENT ASYMPTOTIC EFFICIENCY.
C
C
C       *****ALGORITHM NOTES:
C       THE INPUT PARAMETER, CONST, IS CHECKED TO AVOID UNDERFLOWS AND
C       OVERFLOWS.
C
C       *****REFERENCES:
C       (1)     BEATON,A.E. AND TUKEY,J.W.(1974), TECHNOMETRICS 16,
C          147-192.
C
C       *****HISTORY:
C       ROSEPACK RELEASE 0.3      JUNE 1976
C$      IF (IBM) THEN
CC      IBM 360/370 VERSION
C$      ELSE IF (XEROX) THEN
CC      XEROX VERSION
C$      ELSE IF (UNIVAC) THEN
CC      UNIVAC VERSION
C$      ELSE IF (HIS) THEN
CC      HONEYWELL VERSION
C$      ELSE IF (DEC) THEN
CC      PDP 10 VERSION
C$      ELSE IF (CDC) THEN
CC      CONTROL DATA VERSION
C$      ELSE IF (BGH) THEN
CC      BURROUGHS VERSION
C$      ELSE, 1 CARD
CC      **********  MACHINE VERSION  **********
C$      IF (SINGLE) 1 CARD, 1 CARD
CC      SINGLE PRECISION DECK
CC      DOUBLE PRECISION DECK
C
C       WRITTEN BY NEIL KADEN (NBER / COMPUTER RESEARCH CENTER)
C       JUNE 23, 1975.
C       MODIFIED 4 NOVEMBER 1975 BY N. KADEN
C       MODIFIED 29 OCTOBER 1976 BY DAVID COLEMAN
C
C       *****GENERAL:
C       QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO:
C          SUPPORT STAFF MANAGER
C          COMPUTER RESEARCH CENTER FOR ECONOMICS AND MANAGEMENT SCIENCE
C          NATIONAL BUREAU OF ECONOMIC RESEARCH
C          575 TECHNOLOGY SQUARE
C          CAMBRIDGE, MASS. 02139.
C
C       DEVELOPMENT OF THIS PROGRAM SUPPORTED IN PART BY
C       NATIONAL SCIENCE FOUNDATION GRANT GJ-1154X3 AND
C       NATIONAL SCIENCE FOUNDATION GRANT DCR75-08802
```

```
C      TO NATIONAL BUREAU OF ECONOMIC RESEARCH, INC.                    BIW01110
C                                                                       BIW01120
C                                                                       BIW01130
C      ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::   BIW01140
C      :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::BIW01150
C                                                                       BIW01160
C                                                                       BIW01170
C      ::::::::::  OFLIM IS THE LARGEST POSITIVE FLOATING POINT NUMBER  BIW01180
C$     IF (IBM1) THEN                                                   BIW01190
CC        IBM 360/370: OFLIM = (16.**63)*(1. - 16.**-6)  ::::::::::     BIW01200
C$     ELSE IF (IBM2) THEN                                              BIW01210
CC        IBM 370/360: OFLIM = (16.**63)*(1. - 16.**-14)  ::::::::::    BIW01220
C$     ELSE IF (XEROX) THEN                                             BIW01230
CC        XEROX: OFLIM = (16.**63)*(1. - 16.**-6)  ::::::::::           BIW01240
C$     ELSE IF (UNIVAC) THEN                                            BIW01250
CC        UNIVAC: OFLIM = (2.**127)*(1. - 2.**-27)  ::::::::::          BIW01260
C$     ELSE IF (HIS) THEN                                               BIW01270
CC        HONEYWELL: OFLIM = (2.**127)*(1. - 2.**-27)  ::::::::::       BIW01280
C$     ELSE IF (DEC) THEN                                               BIW01290
CC        PDP 10: OFLIM = (2.**127)*(1. - 2.**-27)  ::::::::::          BIW01300
C$     ELSE IF (CDC) THEN                                               BIW01310
CC        CONTROL DATA: OFLIM = (2.**1022)*(2.**48 - 1.)  ::::::::::    BIW01320
C$     ELSE IF (BGH) THEN                                               BIW01330
CC        BURROUGHS: OFLIM = (8.**62)*(8.**13 - 1.)  ::::::::::         BIW01340
C$     ELSE, 1 CARD                                                     BIW01350
CC     **********  DATA STATEMENT  **********                           BIW01360
C$     DATA OFLIM /SINFP/                                               BIW01370
       DATA OFLIM /Z7FFFFFFFFFFFFFFF/                                   BIW01380
C                                                                       BIW01390
C      ::::::::::  UFETA IS THE SMALLEST POSITIVE FLOATING POINT NUMBER BIW01400
C         S.T. UFETA AND -UFETA CAN BOTH BE REPRESENTED.                BIW01410
C$     IF (IBM) THEN                                                    BIW01420
CC        IBM 360/370: UFETA = 16.**-65  ::::::::::                     BIW01430
C$     ELSE IF (XEROX) THEN                                             BIW01440
CC        XEROX: UFETA = 16.**-65  ::::::::::                           BIW01450
C$     ELSE IF (UNIVAC) THEN                                            BIW01460
CC        UNIVAC: UFETA = 2.**-129  ::::::::::                          BIW01470
C$     ELSE IF (HIS) THEN                                               BIW01480
CC        HONEYWELL: UFETA = (2.**-128)*(2.**-1 + 2.**-27)  ::::::::::  BIW01490
C$     ELSE IF (DEC) THEN                                               BIW01500
CC        PDP 10: UFETA = 2.**-129  ::::::::::                          BIW01510
C$     ELSE IF (CDC) THEN                                               BIW01520
CC        CONTROL DATA: UFETA = 2.**-975  ::::::::::                    BIW01530
C$     ELSE IF (BGH) THEN                                               BIW01540
CC        BURROUGHS: UFETA = 8.**-51  ::::::::::                        BIW01550
C$     ELSE, 1 CARD                                                     BIW01560
CC     **********  DATA STATEMENT  **********                           BIW01570
C$     DATA UFETA /SETA/                                                BIW01580
       DATA UFETA /Z0010000000000000/                                   BIW01590
C                                                                       BIW01600
C                                                                       BIW01610
C      *****BODY OF PROGRAM:                                            BIW01620
       IF (CONST .LE. 1.000) PROD = OFLIM * CONST                       BIW01630
       IF (CONST .GT. 1.000) PROD = UFETA * CONST                       BIW01640
```

```
FILE: BIWGT    FORTRAN  A              CORNELL VM/370 3.4

C     ::::::::::: CONST IS IN RANGE ::::::::::
C                                                                       BIW01660
      DO 100 I=1,N                                                      BIW01670
         U1 = DABS( U(I) )                                              BIW01680
         IF (U1 .LE. CONST) GO TO 10                                    BIW01690
C        ::::::::::: DABS(U(I)) .GT. CONST ::::::::::                   BIW01700
         SOW(I) = 0.0D0                                                 BIW01710
         GO TO 100                                                      BIW01720
   10    CONTINUE                                                       BIW01730
         IF (CONST .GE. 1.0D0) GO TO 20                                 BIW01740
         IF (U1 .LE. PROD) GO TO 20                                     BIW01750
C        ::::::::::: DIVISION WOULD OVERFLOW ::::::::::                 BIW01760
         SOW(I) = 0.0D0                                                 BIW01770
         GO TO 100                                                      BIW01780
   20    CONTINUE                                                       BIW01790
         IF (CONST .LE. 1.0D0) GO TO 30                                 BIW01800
         IF (U1 .GE. PROD) GO TO 30                                     BIW01810
C        ::::::::::: DIVISION WOULD UNDERFLOW ::::::::::                BIW01820
         SOW(I) =  1.0D0                                                BIW01830
         GO TO 100                                                      BIW01840
   30    CONTINUE                                                       BIW01850
C        ::::::::::: FUNCTION CAN BE COMPUTED NORMALLY ::::::::::       BIW01860
         U1 = U(I) / CONST                                              BIW01870
         SOW(I) = ((0.5D0 + U1) + 0.5D0)*((0.5D0 - U1) + 0.5D0)         BIW01880
  100 CONTINUE                                                          BIW01890
C                                                                       BIW01900
      RETURN                                                            BIW01910
C     ::::::::::: LAST CARD OF WBIWGT ::::::::::                        BIW01920
      END                                                               BIW01930
                                                                        BIW01940
```

An Experimental Interactive System for Integer Programming

Monique Guignard

Wharton School

University of Pennsylvania

Kurt Spielberg

IBM Scientific Marketing

Abstract

The paper describes an experimental interactive system for the solution of integer programming problems (primarily of 0-1 nature, but not exclusively so)

The system includes most techniques which appear to offer hope of overcoming the combinatorial difficulties of all integer programming algorithms: in particular LP with cuts, BB with propagation, State enumeration, Interval reduction, Preferred variable reduction, Exploitation of Benders inequalities, Local search, Heuristics, etc..

There is emphasis on interaction by the analyst at the terminal. An example illustrates the use of various tools. A table summarizes results obtained by a number of different approaches to a problem of moderate difficulty.

1. Introduction

We describe an experimental interactive system for 0-1 programming. The programs are written in APL. Various techniques have been or are planned to be incorporated in this system. They are intended to be called by the user, independently from each other (subject to restrictions which we would like to make as little painful as possible) , or sequentially, depending on the results of the experimentation.

No single technique can be expected to solve all types of integer problems. But our preliminary experimental results are encouraging and suggest that a truly flexible interactive system has potential for aiding in the understanding and solution of integer programs, beyond what can be expected from a standard preset program.

The paper is not intended to give all algorithmic details. In section 2 an overview of techniques and features is given, with references to expository papers. For convenience, certain key concepts are briefly summarized in an appendix.

What counts here is that there *are* certain techniques for "looking" at the problem, gleaning more information from it in simple form (e.g, in terms of simple logical relations among variables, simple inequalities, stronger bounds, etc.) by *judicious action at a computer terminal*, guided by the printout of the *"current"* state of the solution process. The user may find some help from the example presented in section 3.

Can interactivity play a major role in solving difficult problems? We increasingly believe that the answer is yes if the user of the system is well versed in integer programming. Whether one shall be able eventually to construct useful interactive system for the non-specialist is somewhat difficult to predict now. We are optimistic, but much work remains to be done.

2. Techniques of the System

At present the following features and techniques have been incorporated :

2.1 Linear Programming (LP),

with possible addition of Gomory-Johnson *cuts* and dual *reoptimization* until either no real progress is made any more (in terms of changing objective function), or until the number of reoptimizations or cuts reaches upper bounds imposed by the user. <1,2,3,4>

2.2 Branch and Bound Programming (BB),

with *propagation* on nonbasic (possibly *preferred*) variables. A single LP is solved at the current origin (or node) of the search tree and *penalties* are computed. A branch on a nonbasic variable set at its optimal LP value is called *propagating*. One will want to propagate as long as the *"alternate"* *penalty* is large, <5,6,7>.

2.3 Benders Inequalities,

which are generated at the end of an executed LP. They are intended to be used either

(i) after solution of a problem within a BB or Enumeration scheme (usually with some aid from cutting planes), or

(ii) after *guesses* at partial integer solutions have been entered by the user in order to get logical conditions which might characterize the problem. After a guess is entered, the remaining linear program is resolved and the Benders inequality is taken from the final LP tableau (whether feasible or infeasible).

Note: it may not be permissible to use cuts in such an instance.

In general, Benders inequalities are collected and *retained* for exploitation in terms of *reduction* (see 2.4, 2.5 below). In contrast, Gomory-Johnson cuts (and we believe other cuts of similar genesis as well) are *not* well suited for reduction (i.e., they lead to logical inequalities of large degree). After use in an LP, they are therefore usually discarded. <8,9>

2.4 Reduction Procedures,

are invoked throughout to
(i) shrink bound intervals , and
(ii) to generate logical inequalities

The interval reduction procedure is used iteratively, screening all constraints so as to compute the tightest possible bounds on structural *and* slack variables. Great care must be taken to avoid round-off difficulties; i.e., one must use appropriate scaling and tolerances (as in LP), especially when Benders inequalities are used for reduction. <9,10,11,12,13>

2.5 Logical Inequalities.

Given any inequality in (0,1) or integer variables, one can derive from it a set (possibly empty, but not usually so) of *minimal preferred variable inequalities (m.p.i.'s)*. The *degree* of the system ("size" of the smallest inequalities; the degree of an inequality i will be called PI(i) in the printouts) is a good measure of the tightness of the problem. The bound interval of one of the preferred variables, at least, must be shrunk by one unit.

These logical inequalities are among the main tools for guiding both the BB scheme and the enumeration (see also "penalty improvement"). <9,10,14,15>

As the referee points out, there exist logical inequalities ("Boolean", "Canonical" inequalities; e.g., see <16,17>) which are more general than m.p.i.'s. They are especially important in the complete characterization of the underlying problem. However, their overabundance may present computational difficulties. We believe we have good reasons for preferring to work with the more special properties of m.p.i.'s (used computationally already in <9>).

2.6 Enumeration,

is based on the *additive algorithm* of Balas <18>, modified in a number of ways, with the search starting at the origin y=(0,0,...,0). Branches are restricted to *minimal preferred variable sets*. The actual set to be used is the *union* of those preferred variables (in minimal preferred sets) which have good *contraction properties*:
1 st priority: double contraction
2 nd priority: single contraction
3 rd priority (default): all free variables.
(a branch is "contracting" when the degree of the system is guaranteed to be decreased as a result. Contraction can be determined from the set of minimal preferred inequalities). Within the finally chosen set, the actual branch variable is selected so as to lead to minimal (maximal) overall infeasibility of the problem at the next node in phase 1 (phase 2) of the enumeration. <10>

In the above context, phase 1 is meant to be the period during which the search is directed towards finding improved feasible solutions, whereas phase 2 is to deal with the establishment of optimality (possibly within certain tolerances) once a good integer solution has been generated.

Note: These branching criteria can be rendered inoperative when the "local search" (see 2.10) identifies an improving direction.

2.7 State Enumeration,

differs from enumeration as follows. At any node, a state is determined (usually by rounding of an LP solution). It is essentially a particular value for the integer vector y, say $\hat{y}$, believed to be close to feasible integer solutions. In the zero-one case, the search variables for an enumeration such as that of 2.6, are then taken to be $y(j)$ if $\hat{y}(j) = 0$ and $1 - y(j)$ if $\hat{y}(j) = 1$. (See <12> for the generalization to the integer case with small bound intervals).

I.e., the final enumeration is carried out with transformed variables $y(j)$ set at 0 initially and increased on forward steps of the algorithm (as in <18>). In _dynamic_ State Enumeration, we can envisage that the search variables $y(j)$ may be redefined over the free variables at each node, provided that reasonably good information is available for doing so.

We have found State Enumeration, with strategies roughly as outlined in 2.6, 2.7 (there are many other possibilities of sometimes highly complex nature), highly effective in finding good feasible integer solutions fast, especially when augmented by a simple _"one-level"_ local search and by a _"ceiling test"_ (see <2.8,2.9>). <3,4,14>

2.8 _Ceiling Test_

Most enumeration schemes have tests involving the objective function of the initial tableau (often with non-negative cost coefficients). In state enumeration, the original objective function becomes less interesting and can be taken care of, anyway, as a special Benders inequality. Instead of the original objective function, we have found it useful to concentrate on the final objective function row from the optimal tableau of an initial LP problem (with cuts added when deemed desirable). The row is retained (coefficients and all necessary information about the nature of the nonbasic variables) and permits the fixing of variables and/or shrinking of bound intervals for the nonbasic variables, among them original slack variables as well as structural variables.
It should be noted that the shrinking of slack bound intervals can be used as input to the various reduction procedures and can lead to further information about all the variables there.

2.9 _Bounds on Slacks; Compression_

Our linear programming system and all related integer programming procedures permit the imposition of upper and lower bounds on the slacks. Equality constraints, for example, are handled by imposition of zero bounds above and below. The minimal inequality reduction procedures work with inequalities only, and we generate two such inequalities as input to reduction whenever we know (or believe) that the upper bounds on the slacks are "true" (i.e., truly restrictive) bounds.

Some of our sample problems are known to have two inequalities representing actually only one equality constraint. The system therefore has a "compress" function detecting such situations and (re)creating the (original) equality constraints.

2.10 _Local Search_

Any BB or Enumeration algorithm may benefit from a search around the "current" point under consideration. In the present system, the "depth" of the local search is controlled by one parameter (LEV, or λ). The value 0 corresponds to no search (just one , the current, point considered), the value 1 corresponds to the alteration of all f "free" (not fixed) variables by one unit at a time. In general, one enumerates the $(f!)/(f-\lambda)!$ adjacent points. In view of the exponential increase in the number of such points, only the values 1 and 2 and conceivably 3 appear to be reasonable for λ .

2.11 _Heuristics for Feasible Solutions_

Several heuristic methods have been programmed which try to generate feasible solutions to a system of linear inequalities in 0-1 variables. One of them starts with a given 0-1 vector and modifies the component which maximally decreases the sum of the infeasibilities or the number of infeasibilities. Such steps are repeated as often as specified. One way of using such modifications is to have a complete "forward pass" over all the variables, followed by a complete "backward pass", i.e. all variables are altered in a pass, the sequence being determined by one of the criteria mentioned above, <19>.

Another heuristic concentrates on the logical inequalities of small degree and tries to generate a feasible solution to the current system of minimal preferred inequalities, which can then be used as a starting point for the first heuristic, or for a local search, or as a state for the state enumeration algorithm. <3,4,14>

2.12 Row Combination

It has been known for a long while that linear combinations of rows may be more interesting than the rows themselves, but there is little insight into how the inequalities are to be generated (possible exceptions are references <20> and <21>). Consistent with the main emphasis of this paper, we have chosen to take the *degree* of an inequality as its basic measure of strength. Hence we allow for "visual" combination of (≤) inequalities (e.g., so as to have coefficients of opposite sign combine to produce small results in magnitude on the left and large negative right hand sides, if possible), and we test the resultant inequality by reduction. Alternatively, we also have heuristic algorithms of modest capability which utilize minimal preferred inequalities to give automatic row combinations, with reduction used again to check out the results and terminate the process.

2.13 Improvement of Penalties

Penalties can be improved by taking into account logical conditions on the integer variables, such as the minimal preferred inequalities. If some nonbasic variable must be modified, it may be the case that at least some of a set of other nonbasic variables must also be altered to achieve feasibility. This usually leads to the accrual of an additional penalty. One may utilize improved penalties *a priori* in a preprocessing of the penalty table for BB programming, or *dynamically* to store nodes with increased alternate penalties in BB with propagation. <6,7>

---

3. An Example of a Terminal Session

1) START1

BM1515 is a (16 by 16) tableau, of the form

$$\begin{bmatrix} Co & C \\ -B & A \end{bmatrix}$$

corresponding to the minimization of $z = Co + C \cdot y$, subject to:

$$A \cdot y \leq B$$

$$0 \leq y \leq 1, \text{ all } y(j) \text{ integer} \quad .$$

It represents a 15 variable problem of moderate difficulty, with 15 constraints and a dense constraint matrix (taken from <22>, problem #13).

```
        START1
ENTER M,N
⎕:
        15 15
ENTER TABLEAU, EXPANDED FORM
⎕:
        ⎕←BM1515
   0    7    4    1    3    4    2    6    2    1    5    1    5    7    2    7
 ¯36    2    5    1    0    3    3    2    6    2    2    5    3    3    7    9
  22   ¯6    5   ¯8   ¯3    0   ¯1   ¯3   ¯8   ¯9    3   ¯8    6   ¯3   ¯8   ¯6
   3    5   ¯6   ¯5   ¯3   ¯8    8   ¯9   ¯2    0    9   ¯1    7    9   ¯9    4
  ¯1   ¯9   ¯5    0    9   ¯1    8   ¯3    9    9    3    0   ¯7    5    4   ¯9
 ¯10    8   ¯7    4    5    9   ¯1    7    1   ¯3    2    0   ¯3   ¯5    9    7
 ¯12    7    5    2    0    6    6    7    6   ¯7   ¯7   ¯1   ¯8   ¯3    9    1
   1   ¯1    3    3    4    1    0    4   ¯1   ¯6    0    8    0    1   ¯5   ¯4
   2   ¯1    2   ¯6   ¯9    0    7   ¯9    9    6   ¯4    5    3    1   ¯3   ¯9
   3    4   ¯6   ¯7   ¯2   ¯2    0   ¯6   ¯6    7   ¯4    0    2    8    0    4
  ¯2    0    0    0    0    0    0    0    0    0   ¯1   ¯1   ¯1   ¯1    1    1
 ¯14   ¯1    3    3   ¯4    5    5    7    8   ¯8    0    0   ¯1   ¯2    4   ¯6
  ¯9    8    0    5    0    2    6    7    1   ¯1    1    4   ¯5   ¯7   ¯8    2
  25   ¯2   ¯7   ¯8    6   ¯2   ¯5   ¯9   ¯2    2   ¯8    1   ¯3    5    6   ¯8
  ¯3    7    9   ¯7   ¯5    1   ¯5   ¯5    4    9    3   ¯4    0    0    7   ¯7
 ¯32    7    1    4   ¯3    0    6   ¯3    7    8    1   ¯6    6    8   ¯3    8
```

2) Initial LP. Preprocessing.

No variable can be fixed by reduction in the initial preprocessing. The continous LP optimum is 14.96 and 5 variables out of 15 are fractional. A Benders inequality is computed from the optimal LP tableau and printed out. (ZSTAR stands for the best objective function value found, or for an upper bound supplied by the user. We use a large value as the default).

One enters a "state" ST1, using a rounded LP solution with rounding parameter $p = .5$, (entries 1..keep y(j), entries 0..use complemented variables 1 - y(j)) .

```
BENDERS INEQUALITY OF TYPE 1
( 5.844 0 ¯6.122 6.741 3.03 0 0 ¯1.121 0 0 0.7512 2.862 8.055
1.963 ¯0.7163 ) × Y ≤ ¯22.92 +ZSTAR

ST1=1 0 0 1 1 1 1 0 1 1 1 1 1 1 0
```

3) Guesses

One is prompted for guesses. One enters indices and corresponding values for a subset of the integer variables (unspecified variables being left unconstrained) and resolves one resulting LP.

The final LP tableau yields a Benders inequality of type 1 if feasible, of type 2 if infeasible.

The function GUESS works with the last Benders inequality.

GUESS 0 chooses the free variables with negative coefficients and sets them to 0. The guess is then likely to be infeasible and a new feasibility condition will be generated (B.I. of type 2).

GUESS 1 chooses the free variables with positive coefficients, and one sets these variables to 1. In this case also one constrains the current B.I. and tries to obtain tighter feasibility conditions on a subset of the variables.

Any given guess leads to the resolution of one linear program. While the Benders inequality is retained for further processing (possibly after a check as to whether it is strong enough, i.e. has degree low enough). For large problems, the guesses will most likely be generated automatically , according to a scheme which the user may have developed for a smaller problem of similar structure.

Having some familiarity with a given problem (and by that we do not mean knowing its solution), one can usually think of a number of other promising guesses. E.g., one may want to set to one some variable(s) with large positive coeffient(s) or at zero some variable(s) with large negative coefficients; or one makes guesses in consonance with one's knowledge of any possible special structure.

```
WANT TO GUESS AT SOLUTION?
YES
ENTER GUESS FOR Y, FIRST INDICES, THEN VALUES
⎕:
      GUESS 1
2 10 12
⎕:
      1
ACTUAL NUMBER OF ROWS FOR MAT :18
PROBLEM NOT FEASIBLE
DETERMINANT=   1.004E6
BENDERS INEQUALITY OF TYPE 2
( 36 ¯1.563 ¯10.11 0 11.02 5.74 0 0 ¯6.753 25.63 0 6.149 0 0 0 )
× Y ≤ 12.12
```

```
WANT TO GUESS AT SOLUTION?
YES
ENTER GUESS FOR Y, FIRST INDICES, THEN VALUES
⎕:
      7
⎕:
      0


11.47  SEC
OBJECTIVE FUNCTION =  15.12
NO. OF PIVOT STEPS = 29
DETERMINANT=   1.045E6
STRUCTURALS: 0.01126 0.7627 1 0 0 0.0874 0 1 0.3813 0.2613 0.1264
 0 0 0 1
BENDERS INEQUALITY OF TYPE 1
( 0 0 ¯9.671 8.708 2.266 0 ¯3.909 ¯0.6645 0 0 0 3.872 11.21 2.857
 ¯6.548 ) × Y ≤ ¯32 +ZSTAR
```

4) After all guesses have been entered, and the corresponding LP's have been solved, with a final Benders inequality always adjoined to the tableau, PREPROC checks for interval reduction and fixing of variables. In this problem variables y1, y9 and y15 can be fixed at 0, 1 and 1, respectively.

```
N. VARS. FIXED 3

          LU2[1;]        (lower bounds)
0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
          LU2[2;]        (upper bounds)
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
↑               ↑           ↑
y1              y9          y15
```

5) The starting LP for the BB (or an Enumeration) procedure is then solved with these values substituted, and the initial objective function value is thus raised to 15.7. Dual reoptimization after the generation and temporary addition of Gomory-Johnson cuts finally brings the initial BB objective function to 18.39.

```
OBJECTIVE FUNCTION =  15.7
NO. OF PIVOT STEPS = 25
DETERMINANT=   4.967E5
STRUCTURALS: 0 0.8451 1 0 0 0.1926 0.1758 0.3951 1 0.2186 0 0 0 0
 1
BENDERS INEQUALITY OF TYPE 1
( 7.362 0 ¯4.894 5.722 1.515 0 0 0 3.848 0 0 1.463 8.021 3.042
1.645 ) × Y ≤ ¯15.1 +ZSTAR
5  CUT(S) GENERATED


OBJECTIVE FUNCTION =  18.39
NO. OF PIVOT STEPS = 87
BENDERS INEQUALITY OF TYPE 1
( 10.25 0 ¯3.911 3.535 0.09996 0.06115 0 0 4.343 0 0 0 5.384
0.004794 1.39 ) × Y ≤ ¯16.57 +ZSTAR
STRUCTURALS: 0 0.1975 1 0 0 0 0.6106 0.4917 1 0.1856 0.101 0.5849
 0 0 1
```

6) BB starts. The system computes penalties (added to the current objective function value to give a maximal lower bound on the final objective function value).

In the normal BB mode, the generated node (i.e., the current bounds and z + penalty) is stored in the node table, and the next node would be brought in.

In the propagation mode one looks for variables to be fixed at their current value, with an alternate node stored in the table, and the current node being processed further.

In this example a propagating variable y(13) is identified and the alternate (y(13)=1; often other conditions can be imposed from the minimal preferred inequalities) is stored with penalty 5.384.

The user is asked whether he accepts the propagation. If the answer is yes, the propagation proceeds; if not, the user has the choice of proceeding in the normal BB mode or of trying to finish the current problem (in core) by state enumeration. In the present case we accept the first propagation (storing an alternate node with penalty 5.384) and reject the second possibility with alternate penalty 3.911 (the system, of course, generates the penalties in descending order of magnitude).

```
PRP.PEN.CTR 13 5.384 0    (CTR =0 ↔ NO CONTRACTION)
                          ( PRP=13 ↔ PROPAGATE  WITH Y13)
ACCEPT      ?             ( PEN= 5.394 ↔ WITH  ALTERNATE
YES                                        PENALTY  PEN )

OUT:        (STORE NODE)
************************
1 23.77 5.384 ¯13  (NODE NUMBER, OBJ. F. VALUE, PEN., BRANCH)


PRP.PEN.CTR ¯3 3.911 0

ACCEPT      ?
NO
FOR ALTERNATE      CALL STRAT, BP, NBP HERE
SAVE NOW, BEFORE ENUM.
```

7) Enumeration starts (from the state ST1). Variable 13 is at 0 (via propagation), and y1, y9, y13 have been fixed globally. One branches on y8=1, and a feasible solution is found immediately (undoubtedly due to the imposition of the state), with obj function z=26.

```
PI 4 4 4 2 2 4 3 2 3 10 5 3 4 3 4 10 4 2 2 2 10 10 2 10 3 2 10 5 (rows degrees π(i)
K IS 1   (local  search level)
Z      26
 FEASIBLE SOLUTION 26
SOLUTION FOUND AT NODE  2 LEVEL 1 PHASE 1
**************************************************
OBJECTIVE FUNCTION 26
NEW ZSTAR =26
SOLUTION 0 1 1 0 0 0 1 1 1 0 0 1 0 0 1
INT. SOL. FEASIBLE  , Z= 26
```

8) Enumeration is completed after examination of a total of 17 nodes.

```
L/BND= ¯1  INF. IN BOUNDRD  (IN BOUND REDUCTION)
END OF ENUMERATION
******************
OPTIMAL SOLUTION 0 1 1 0 0 0 1 1 1 0 0 1 0 0 1
OPT. OBJ. FUNCTION 26
```

9) Return to BB.

The stored alternate node is brought back from the tree into high speed storage. After imposition of the bounds, the inequality system (including Benders inequalities) with $z \leq z*\leq 1= 26-1 = 25$, is now infeasible. End of BB.

Termination of integer program.

```
IN OF NODE 1
*******************
1 23.77 5.384 ¯13  (NODE NUMBER, OBJ.F.VALUE, PEN., BRANCH)


RED       (REDUCTION)
NOPV 0  (π OF SYSTEM)
ρPLUS 20 15
INFEASIBLE;NOPV      0
```

4. Computational Results

There are a number of problems which can be entirely resolved by the use of one or the other technique. For example, a (28,35) problem of the tanker scheduling variety (see <9>) yields an integer solution after imposition of a total of 6 Gomory-Johnson cuts in 2 reoptimizations.

The same problem yields well to state enumeration techniques (11 BB nodes of the propagation type, i.e. with only one true linear program resolved, followed by between 9 and 33 enumeration nodes depending on the initial state; at the end none of the alternate BB nodes need to be processed further).

On the other hand it requires a large number of LP optimizations in a straightforward BB optimization run (between 40 and 109 linear programs depending on strategy).

In the following table we describe various approaches to the (15 by 15) problem used as example in the text. In some respects it is rather difficult: the gap between LP obj. function and integer solution objective function is large; the penalties are rather small ; the response to Gomory-Johnson cuts is only moderate; the degree of the initial system is 3. It behaves much better when Benders inequalities are added, permitting the reduction of the degree to 1 (fixing of three variables); etc..

In the table we summarize a number of runs using some or all of the available tools. One does or does not add Gomory-Johnson cuts. One does or does not enumerate at a selected pending node of the BB tree. The state for enumeration can be chosen arbitrarily: we ran the program with the state determined from the rounded LP solution, with all fractional variables less than or equal to .5 fixed at 0, and all variables larger than .5 fixed at 1.

We entered some guesses at a solution and appended the resulting Benders inequalities to the original tableau.

Which of the results are considered to be best will depend on the relative computational efforts, and therefore on the computer and on the system used (APL in our case). In general, we feel it most essential to reduce the number of linear programs, but even more fundamentally, to avoid large trees (many BB nodes, large enumeration levels), i.e. situations in which the combinatorics overwhelms the problem solver.

The interactive system appears to give the insight and often the tools for avoiding exponentially intractable situations; i.e., even for more difficult problems (e.g., the (37 by 74 problem of <9>), which we did not run to conclusion because of slow APL response at the terminal, it was clear that the search was "well-behaved", the tree remaining a "low-level tree" , with steady progress being made .

Initial LP value	14.96	14.96	14.96	14.96	14.96	14.96
Total n. of cuts	0	0	0	47	35	41
reoptimizations	0	0	0	7	7	7
Final value of LP	14.96	14.96	14.96	16.63	16.53	18.39
Integer optimum	26	26	26	26	25	26
Use B-B?	no	yes	yes	yes	yes	yes
N. of nodes generated		13	2	2	2	2
.... called		11	2	2	2	2
Optimum found at node		12				
Use enumeration?	yes	no	yes	yes	yes	yes
State for enumeration						
from LP?			yes	yes	yes	yes
1?	yes					
0?						
N. of nodes	105		63	37	19	17
Optimum found at node	73		31	25	10	2
Use guesses?	no	no	no	yes	yes	yes
Type of guesses				random		
N. of guesses				10	6	10
N. of Benders Ineq.				12	13	12
N. of variables fixed				0	1	3
N. of LP solved	0	25	2	9	7	10

Table

Six Approaches to Solving the (15,15) Sample Problem

5. Appendix: Definition of Some Terms <10>

. A *logical* inequality in bivalent variables is meant to be an inequality with 0, 1, -1 coefficients only.

. A *preferred variable* inequality (abbreviated p.i.) is a logical inequality of the form $\sum_{j\in J}\tilde{y}(j) \geq 1$, $\tilde{y}(j)$ being either $y(j)$ or $1 - y(j)$, which expresses the condition that at least one of the $\tilde{y}(j)$, $j \in J$, must be one.

. The values $\tilde{y}(j) = 1$ (i.e., either $y(j)=0$ or $y(j)=1$, depending on the nature of the $\tilde{y}(j)$) are termed the *suggested* or *indicated* values of the preferred variables.

. Given some procedure for generating a set of preferred inequalities, we call *minimal* (relative to the procedure) those p.i.'s with minimal number of non-zero coefficients.

. The *degree* d of a m.p.i. system (and by extension the degree of the initial inequality or system of inequalities from which the m.p.i. system was derived) is the number of non-zero coefficients of one of the minimal preferred inequalities.

6. References

<1> Gomory, R.E., and E.L. Johnson, "Some Continuous Functions Related to Corner Polyhedra" , 1. Math. Progr., Vol. 3, #1, 1972; 2. Math. Progr., Vol 3, #3, 1972

<2> Guignard, M., "Inegalites Valides de Gomory-Johnson", Journees de Combinatoire de l'AFCET, Dec. 1971

<3> Guignard, M., and K. Spielberg, "The State Enumeration Method for Mixed Zero-One Programming", IBM Phila. Sci. C. Rep. 320-3000, Feb. 1971

<4> Guignard, M., and K. Spielberg, "A Realization of the State Enumeration Procedure", IBM Phila. Sci. C. Report 320-3025, June 1973

<5> Land, A.H., and A.G. Doig, "An Automatic Method for Solving Discrete Programming Problems", Econometrica, Vol. 28, 1960

<6> Spielberg, K., "A Minimal Inequality Branch-Bound Method", IBM Phila. Sci. C. Report 320-3024, June 1973

<7> Guignard, M., "Preferred Shadow Prices in 0-1 Programming", Res. Report, Dept. of Stat & OR, Wharton School, Univ. of PA, 1974

<8> Benders, J.F., "Partitioning Procedures for Solving Mixed-Variables Programming Problems", Numerische Mathematik, 1962

<9> Lemke, C.E., and K. Spielberg, "Direct Search Zero-One and Mixed-Integer Programming", J. ORSA, 1967

<10> Spielberg, K., "Minimal Preferred Variable Reduction for 0-1 Programming" , IBM Phila. Sci. C. Rep. 320-3013, 1972

<11> Zionts, S., "Generalized Implicit Enumeration using Bounds on Variables for Solving Linear Programs with Zero-One Variables", Naval Res. Logistics Quarterly, 1972

<12> Guignard, M., and K. Spielberg, "Reduction Methods for State Enumeration Integer Programming", Workshop on Integer Programming, Bonn 1975 (to appear in Discrete Mathematics)

<13> Guignard, M., and K. Spielberg, "Zero-One Zero-One Programming and Enumeration" , in "Nonlinear Programming Vol. 2", ed: O. Mangasarian, R.R. Meyer, S.M. Robinson , Acad. Press, 1975

<14> Guignard, M., and K. Spielberg, "Search Techniques with Adaptive Features for Certain Integer and Mixed-Integer Programming Problems", Proceedings of the V th IFIPS Conference, Edinburgh, 1968

<15> Guignard, M., "Contraintes Additionelles en Variables Bivalentes", Pub. Labo. Calcul, Univ. of Lille, Report #20, 1970

<16> Hammer, P.L., "Boolean Procedures for Bivalent Programming", Univ. of Waterloo, Dept. of Combinatorics and Optimization Reports, CORR #73-1, Oct. 1973 (with numerous references to earlier related work going back to the 60's, some of it jointly with authors such as S. Rudeanu and F. Granot)

<17> Balas, E., and R. Jeroslow, "Canonical Cuts of the Unit Hypercube", SIAM Journal of Applied Mathematics, Vol. 23 (1), July 1972 (work dating back to Carnegie Mellon University Report #198 of 1969)

<18> Balas, E., "An Additive Algorithm for Solving LP with Zero-One Variables", J. ORSA, Vol. 13, 1965

<19> Guignard, M., "Methodes Heuristiques de Resolution d'un Systeme d'Inegalites en Variables Entieres ou Bivalentes", Pub. Labo. de Calcul, Univ. of Lille, Report #32, 1972

<20> Bradley, G., "Equivalent Integer Programs and Canonical Problems", Management Science, 1971

<21> Hammer, P.L., Padberg, M.W., and U.N. Peled, "Constraint Pairing in Integer Programming", Univ. of Waterloo, Dept. of Combinatorics and Optimization Res. Reports, CORR #73-7, Aug. 1973

<22> Bouvier, B., and G. Messoumian, "Programmes Lineaires en Variables Bivalentes", Thesis, Faculte des Sciences, Univ. of Grenoble, 1965

AN ACCELERATED TECHNIQUE FOR RIDGE FOLLOWING USING CONJUGATE DIRECTIONS*

Edwin H. Neave
Queen's University, Kingston, Ontario

and

Timothy L. Shaftel
The University of Arizona, Tucson, Arizona and
Queen's University, Kingston, Ontario

ABSTRACT

In this paper we study an accelerated version of Powell's conjugate direction technique for solving unconstrained nonlinear problems. This technique employs a method of taking large steps to enhance movement along ridges. Not only does this technique improve the speed of convergence for nonquadratic problems, but it improves the robustness of the procedure. An APL code, useful for research because of its interactive capabilities, is presented and described in detail.

I INTRODUCTION

The topic of nonlinear unconstrained problem solving is by no means new to the OR/MS literature. Indeed, many of the basic concepts of solving this type of problem have been known for some time. Beginning approximately in 1960 there has been a proliferation of papers on unconstrained optimization techniques. Of these, one of the best known and most extensively tested is Powell's method of conjugate directions. This paper studies Powell's technique and offers a modification that our experiments show improves the rapidity of convergence to a solution when the criterion function exhibits nonlinear ridges. It employs a code that displays the robustness of the technique as well as effecting our particular means of accelerating convergence. The former is demonstrated by a discussion of the effects on program operation of changing various computational parameters. We also demonstrate the usefulness of interactive languages, APL in particular, for this kind of research.

Powell's (1965) conjugate direction method is chosen for several reasons. Among other tests of Powell's method, Box's (1966) paper indicates that it is competitive with several other methods on a variety of nonlinear problems. Additional discussions of conjugate gradients can be found in Fletcher and Reeves (1964), Powell (1965) and more recently Zangwill (1969). Moreover, Powell's technique is straightforward and can be implemented rather easily without requiring either large amounts of core storage or extensive manipulation of stored variables, so that it constitutes a reasonable choice for solving large scale problems. Another reason for the selection of Powell's technique is that it avoids the necessity of calculating derivatives. Finally, the conjugate direction technique has the property of finite convergence for quadratic functions; in this sense it is similar to other techniques including those by Fletcher and Powell (1963) and Fletcher and Reeves (1964), although the latter two cases make use of explicit calculations of the first derivative.

---

*Queen's University, Kingston, Ontario Working Paper No. 76-4, May 1976.

The outline of the paper is as follows. In Section II we briefly discuss Powell's algorithm and the minor modifications we made in setting up our comparison standards. In Section III we examine the ridge following problem and our method of accelerating the convergence of Powell's technique when these ridges are nonlinear. The results of tests on several example problems are reported. In section IV we discuss some problems caused by variation in certain input parameters. Section V presents our position regarding the usefulness of APL for carrying out the type of investigation reported here. Finally Section VI presents our conclusions. The APL listing of the code that generated our results is included in an Appendix.

II THE TECHNIQUE

In this section we present only enough of Powell's method to be able to identify the differences between Powell's original code and the version we have employed. The major difference, leading to accelerated convergence in problems with nonlinear ridges, is permitting Powell's algorithm to commence a new iteration at a point different from (and usually worse in terms of function value) the point at which the previous iteration ended. This modification, which we call a leap, is discussed fully in Section III. In order to demonstrate the comparability of our results, it is important to indicate the additional, minor changes to Powell's technique that we have found useful. Essentially, these minor modifications are designed to permit making certain choices of computational parameters that we employed in developing the accelerated convergence procedure. Powell's basic code may be expressed in the following steps.

i) For r = 1 to n calculate a minimum along direction d_r beginning at the resulting minimum of the last direction searched. The initial point is arbitrary while the starting directions are the coordinate directions of the domain of the search. For any iteration let p_o be the first point and p_n the final point.

ii) If this is the first iteration, recompute the one dimensional minimizations using the initial set of directions. Otherwise, calculate a new direction from $(p_o - p_n)$ and minimize the objective function in this direction. The resulting point is p_o for the next iteration.

iii) Of the r + 1 directions select one to be removed by the criteria given below and return to i).

Our modifications to Powell's method involve the manner in which directions are removed, choices of some computational parameters, and convergence criteria.

Criteria for Removal of Directions. Powell presents a scheme for determining which direction should be removed, a scheme that includes the maintenance of the original directions. Our criteria merely removes (from the set of r original directions) that one which is most nearly parallel to the new one (i.e., the smallest angle between the two normalized direction vectors).

Tolerance Parameters. We use the term "tolerance parameters" to refer to those parameters which the operator must choose before running the program. Powell's technique employs well defined rules for modifying these parameters during the operation of the code, but the tolerance parameters themselves are rarely specified in any discussion of the tests of proposed codes (in Powell's work or elsewhere). We believe that in most instances robustness of a particular technique is an important feature of its operation, since if particular parameters must be determined before a given routine will work satisfactorily it can imply a great deal of testing on particular functions before a good selection becomes possible. In duplicating Powell's technique we experienced difficulty in determining the exact computational parameters to employ, but partially overcame the difficulty through using the interactive properties of APL to obtain comparisons with Powell's reported test results on an iteration by iteration basis.

Five different parameters were specified for the operation of our program. In the listing of the program in the Appendix the 3 - vector TOL is used to specify the first three of these parameters. These values are also used by Powell and are termed m, q and ϵ in his paper. The last two parameters we employ are unique to our code and are called MPASS and MTEST respectively. All five parameters are used during the operation of the one-dimensional search technique used in our procedure. A brief description of these parameters follows:

TOL [1] is the upper bound on the length of a step taken by the one dimensional search technique. It is invoked whenever a quadratic prediction is further away from the points used for the prediction than the value TOL [1] permits.

TOL [2] is the magnitude of the step

length along the direction being minimized. This value is used to determine the position of two additional points needed for the first quadratic fit.

TOL [3] is the required accuracy to which variables along this line must be determined before the search for a minimum will be terminated (unless other termination criteria are invoked). In the operation of our algorithm each of the above parameters affects the search dynamically in that they are employed in conjunction with the direction vector currently used. The end result is that tighter bounds are implicitly invoked as the movements of the variables become smaller.

MPASS is the maximum number of function evaluations allowed during any normal linear search. Our use of this parameter resulted from the discovery that the number of quadratic fits used to predict a particular minimum along a line was a more useful parameter than TOL [3]. We find that the number of iterations is much easier to control directly rather than with the use of TOL [3]. Moreover, by determining the number of quadratic fits we use for any iteration, the accuracy of the search can be controlled in any event, as we shall show.

MTEST is used in an attempt to improve the robustness of the program. In the program we allow for a reduction of TOL [2] by a factor of ten whenever TOL [1] is invoked or when a predicted minimum turns out to be larger than any of the three predicting points. The result is that within a single linear search TOL [2] is scaled down if unreasonable predictions have been achieved. MTEST limits the number of times this reduction can occur for any one linear search.

STOPPING RULES. No specific routine was written to stop our code, so that the result is a termination by default whenever no new minimum is generated in any conjugate direction. Stopping under these conditions is a result of machine accuracy. It would be an easy matter either to invoke stopping under less stringent criteria or to use Powell's stopping rules. Because of the nature of our research, however, we chose merely to input a specified number of iterations and include the capability of restarting the program without loss of previous information. This procedure might also prove practically useful during batch processing. Especially, the ability to stop the operation and perform some cost-benefit analysis before deciding whether to continue the search might be useful when the nonlinear program is working as a subroutine to a more general program.

Duplication of Powell's Results. Table I displays the results of using our version of Powell's code on the three test problems reported in his paper. The results of our code use TOL = 10, .1, .001; MPASS = 4 and MTEST = 1. The similarity in number of evaluations is, of course, not surprising, and is included only to demonstrate the similarity in our procedures as used so far. In the rest of the paper comparison standards will be the results obtained using this version of Powell's code. These results will be compared with those obtained from using the leapsize modification we have developed to accelerate its convergence. The nature of this modification is discussed next.

III AN ACCELERATION TECHNIQUE FOR THE CONJUGATE DIRECTIONS SEARCH

A well known phenomenon in nonlinear search is often referred to as ridge following. In many problems the search begins to follow a long narrow ridge (or valley in the case of minimization) with the result that the program runs for an inordinate length of time while improving the criterion values very slowly. The possibility that changes occur so slowly that the program terminates far from an optimal (global or local) solution is also present in these cases. While conjugate directions work well in the case of linear or near linear ridges, when the ridge curves conjugate directions can tend to zig-zag considerably, much as gradient directions perform in the case of quadratic functions. Figure I provides an hypothetical example of a typical search pattern in this case. The modification of Powell's technique that we report herein was predicated on the belief that it might be useful to continue to move in the general direction of the curvature of the ridge for some distance past the exact minimum along the search direction. We conjectured that this procedure will lead to a result somewhat like Figure Ib, (and thus improve convergence) whenever the function is not too closely quadratic. In these circumstances, the ridges become nonlinear and would, we expected, slow down convergence of the unmodified conjugate directions technique. These conjectures were validated: Figure II provides some actual results of a search we conducted on the Rosenbrock problem. The results of this and other searches are discussed below; supporting data for Figure II appear in Table II (Leapsize = 1 and 3).

In our modification to Powell's code we move in the general direction of the ridge's curvature at the end of each iteration. At this time a new conjugate direction has been generated and we have minimized in this direction. Our modification proceeds by continuing along

PROBLEM	PROGRAM	ITERATION	NUMBER OF FUNCTION EVALUATIONS	FUNCTION VALUE
ROSENBROCK	POWELL	12	142	6×10^{-5}
		13	151	7×10^{-10}
ROSENBROCK	OURS	12	141	2×10^{-5}
		13	151	3×10^{-7}
POWELL (1962)	POWELL	8	126	3×10^{-4}
		10	148	8×10^{-5}
POWELL (1962)	OURS	8	136	4×10^{-5}
		10	163	2×10^{-5}
POWELL (1964)	POWELL	5	not reported	2.9731
(function of 3 variables)		6	not reported	3.0000
POWELL (1964)	OURS	5	102	2.9998
(function of 3 variables)		6	113	3.0000

TABLE I: Comparison of Two Programs to Demonstrate Similar Results. All problems taken from Powell (1964).

---

*Our number of evaluations is adjusted to reflect the fact that we made no use of the unit second derivative as Powell does. This leads to one less evaluation along any direction previously used.

this direction some number of units further than the distance we have just moved. For instance, if in minimizing along the new direction we had moved a distance of one-half unit, at the end of the iteration we take what we term a leap to a new point in that direction (usually with a worse function value). The length of the leap is called a leapsize and indicates how many units we use as a factor to multiply the current direction vector to move between iterations.* The effect of using this parameter is dynamic in the sense that it depends on the size of the direction vectors calculated in a given iteration. Thus, as the search gets closer to a minimum the size of leap actually taken reduces to zero.

Table II reports the results of our modification on Rosenbrock's (1960) two dimensional problem:

$$\text{Minimize } f(x_1,x_2) = 100(x_2-x_1^2)^2 + (1-x_1)^2.$$

This function is of interest because it has the type of nonlinear valley we have been discussing. In Table II we compare the results of using Powell's method with the modifications resulting from taking progressively larger leapsizes. It can be seen that Powell's unmodified search technique moves rather slowly along the nonlinear valley of the Rosenbrock function while as expected, the use of the leap moves the search more rapidly along the valley, thus in general effecting a more rapid convergence to a minimum.

Tables III and IV display the effect of leapsizes for a three and a four variable problem respectively on which test results are also reported by Powell (1964). The three dimensional problem is:

$$\text{Maximize } f = \frac{1}{1+(x-y)^2} + \sin(1/2\pi xyz) + \exp\left(-\left(\frac{x+z}{y}-2\right)^2\right)$$

while the four dimensional problem is of the form:

*In our program, when the leapsize parameter is set to 1, the search technique duplicates Powell's. If leapsize > 1, each new iteration begins at a point different from that chosen by Powell's method.

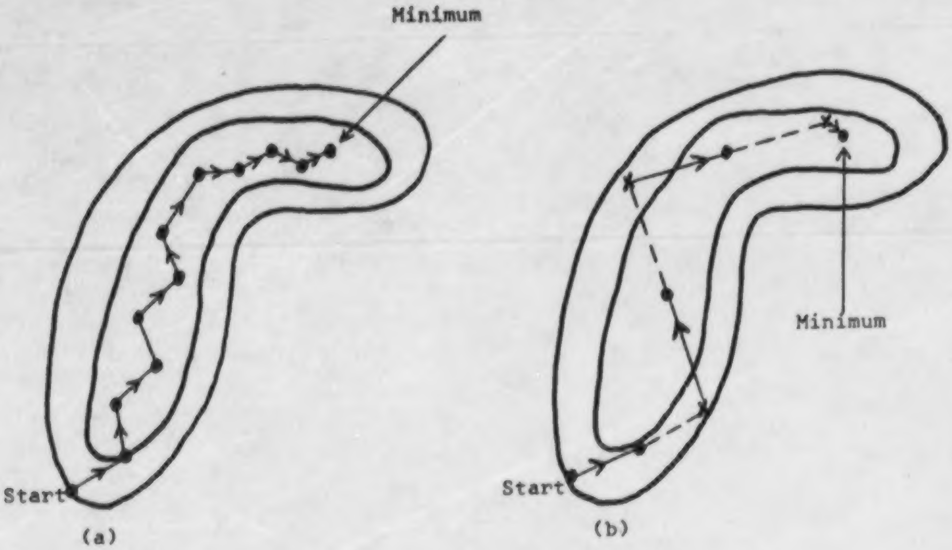


FIGURE I: Possible Improvement in Convergence (a) Following the ridge; (b) Moving away from the ridge.

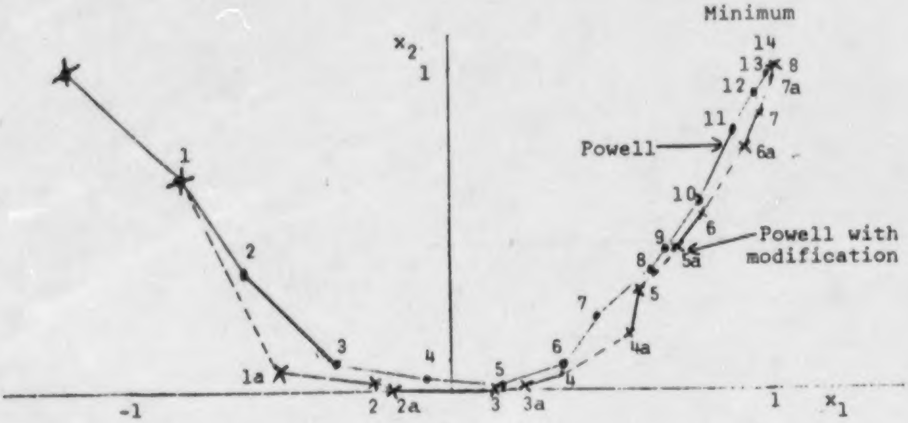


FIGURE II: Convergence of Powell's Technique and its Modification to Solution of Rosenbrock Test Problem. Path chosen by Powell indicated by dots; modification by crosses. Dashed line indicates continuation of movement made by the modification.

	LEAPSIZE = 1 (Powell)				LEAPSIZE = 2*				LEAPSIZE = 3*			
ITERATION	NUMBER OF FUNCTION EVALUATIONS	x_1	x_2	FUNCTION VALUE	NUMBER OF FUNCTION EVALUATIONS	x_1	x_2	FUNCTION VALUE	NUMBER OF FUNCTION EVALUATIONS	x_1	x_2	FUNCTION VALUE
0	1	-1.200	1.000	24.200	1	-1.200	1.000	24.200	1	-1.200	1.000	24.200
1	23	- .837	.677	3.435	23	- .837 (- .684	.677 .372	3.435 3.750)	23	- .837 (-.531	.677 .067	3.435 6.922)
2	37	- .661	.386	3.019	36	- .495 (- .434	.206 .122	2.387 2.494)	36	-.239 (-.199	.037 -.006	1.576 1.645)
3	49	- .372	.098	2.040	49	- .177 (- .082	- .003 -.066	1.508 1.703)	49	.124 (.223	-.008 -.020	.822 1.096)
4	61	- .090	- .027	1.310	62	.080 (.166	- .028 -.037	.967 1.107)	62	.319 (.560	.077 .182	.526 1.941)
5	73	.168	- .002	.783	75	.308 (.475	.068 .138	.553 1.038)	75	.572 (.697	.308 .423	.218 .485)
6	85	.352	.088	.546	88	.525 (.613	.254 .329	.272 .362)	88	.759 (.907	.564 .765	.075 .346)
7	97	.462	.222	.298	101	.716 (.819	.496 .626	.110 .235)	101	.929 (1.000	.857 .980	.008 .039)
8	106	.608	.382	.168	114	.890 (.967	.782 .909	.020 .062)	114	1.000	1.000	1×10^{-6}
9	117	.660	.427	.122	127	.995 (1.028	.989 1.054	2×10^{-4} .002)				
10	131	.783	.599	.067	140	1.002 (.993	1.003 .983	5×10^{-6} 4×10^{-4})				
11	143	.895	.802	.011	153	1.000	1.000	1×10^{-7}				
12	155	.959	.916	.003								
13	167	.996	.991	2×10^{-5}								
14	179	.999	.998	3×10^{-6}								
15	191	1.000	1.000	2×10^{-11}								

Table II: Solution of Rosenbrock's Function with various Leapsizes
TOL = 10, .1, .0001, MPASS = 4, MTEST = 1
*Points in parentheses indicate the beginning of the next iteration.

ITERATION	LEAPSIZE = 1 NUMBER OF FUNCTION EVALUATIONS	LEAPSIZE = 1 FUNCTION VALUE	LEAPSIZE = 2 NUMBER OF FUNCTION EVALUATIONS	LEAPSIZE = 2 FUNCTION VALUE	LEAPSIZE = 4 NUMBER OF FUNCTION EVALUATION	LEAPSIZE = 4 FUNCTION VALUE
1	40	2.764	41	2.639	39	1.994
2	59	2.859	58	2.640	56	2.629
3	73	2.888	74	2.681	73	2.728
4	88	2.965	90	2.715	89	2.729
5	104	2.983	109	2.976	109	2.825
6	120	3.000	125	2.999	128	2.995
7			141	3.000	143	2.999
					158	3.000

TALBE III: Solution of Three Variable Problem with various Leapsizes TOL = 10, .1, .0001
MPASS = 4
MTEST = 1

ITERATION	LEAPSIZE = 1 NUMBER OF FUNCTION EVALUATIONS	LEAPSIZE = 1 FUNCTION VALUE	LEAPSIZE = 2 NUMBER OF FUNCTION EVALUATIONS	LEAPSIZE = 2 FUNCTION VALUE
2	54	.2040	55	.1401
4	91	.0075	92	.0021
6	129	.0018	133	.0018
8	168	4×10^{-5}	172	6×10^{-5}
10	203	2×10^{-5}	209	5×10^{-5}

ITERATION	LEAPSIZE = 4 NUMBER OF FUNCTION EVALUATIONS	LEAPSIZE = 4 FUNCTION VALUE	LEAPSIZE = 8 NUMBER OF FUNCTION EVALUATIONS	LEAPSIZE = 8 FUNCTION VALUE
2	55	.1064	56	.1141
4	95	.0012	97	9806.
6	135	.0003		
8	174	4×10^{-6}		

TABLE IV: Solution of Four Variable Problem with Various Leapsizes
TOL = 10, .1, .0001
MPASS = 4
MTEST = 1

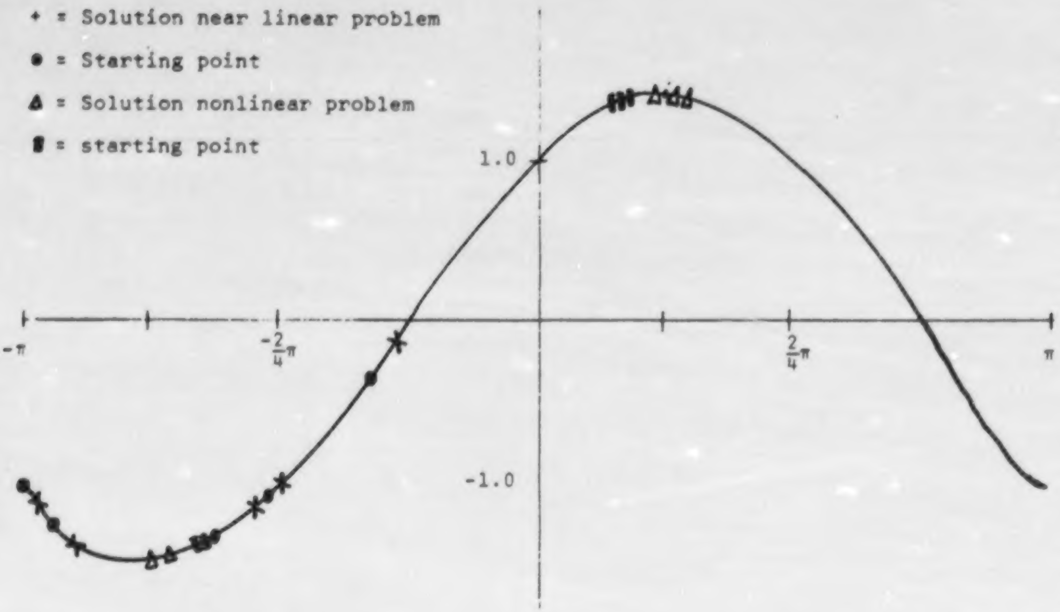


FIGURE III: SIN X + COS X

ITERATION	NUMBER OF FUNCTION EVALUATIONS	FUNCTION VALUE
1	45	153.384
2	69	82.772
3	93	32.300
4	117	15.084
5	141	8.190
6	164	.285
7	186	.005
8	206	.003
9	226	3×10^{-3}
10	244	3×10^{-9}

TABLE V: Solution of Random Trigonometric Problem Using Powell's Method

TOL = 10, .1, .0001
MPASS = 4
MTEST = 1

$$\text{Minimize } f = (x_1+10x_2)^2 + 5(x_3-x_4)^2 + (x_2-2x_3)^4 + 10(x_1-x_4)^4$$

Inspection of the tables shows that the same leapsize is not necessarily the best for every problem. In Table IV a leapsize of eight moves the point so far from the ridge that it cannot be attained again. We would obviously wish to lower leapsize whenever the final point in an iteration (before the leap) is worse than that of the previous iteration. This will, of course, assure convergence if a local minimum is also a global minimum.

Returning now to the three dimensional results reported in Table III, we see that leaps do not improve the rate of convergence in this case. We interpret this result as stemming from the fact that this particular function approximates a quadratic in the region we are searching. Supporting this belief is the fact that the first complete set of conjugate directions is generated at iteration five, whereupon the search converges immediately to the optimum.

To justify our interpretation, we show the case of quadratic convergence more dramatically for a function which is not obviously quadratic. To effect this demonstration we use the bounded trigonometric function presented by Fletcher and Powell (1963). In this case the function is:

$$\text{Minimize } f = \sum_{i=1}^{n} \{ \sum_{j=1}^{n} (A_{ij} \sin x_j + B_{ij} \cos x_j) - E_j \}^2$$

where the A and B matrices are random numbers between zero and 100 and the solutions x_j are random variables between $-\pi$ and π. The values of E_j are then determined so that each term will equal zero at the optimum. The starting point for each variable is randomized between $+ .1\pi$ and $- .1\pi$ of the optimum solution.

Although the problem is seemingly nonquadratic, it turns out to be difficult to generate a random problem which is not essentially quadratic. Figure III shows the function $\sin x_j + \cos x_j$ and the values of x_j for what we found as a typically generated random 5 dimensional problem. (Results for this problem are reported in Table V.) Note that all these values fall in the linear portion of the curve. Although the values of A and B have some effect on the shape of this curve the fact that there are five such terms in each squared term leads us to believe that the nonlinear effects can be balanced against each other. The resulting function is then approximately the square of five linear terms - a quadratic. We would expect that in a conjugate search procedure the optimum would be found as soon as a complete set of conjugate directions were found. As can be seen in Table V this is exactly the case. At the end of the sixth iteration a complete set of five conjugate directions has been found and used - at this point the procedure moves nearly instantaneously to the optimum. No leaps are displayed for this function since they are, of course, not going to improve matters any. We might conclude from this experience that perhaps leaps should be ignored until after the first set of conjugate directions has been developed. The study of this problem is interesting particularly in light of its considerable use as a test function for nonlinear problems. The results of these tests will tend to be biased in favor of quadratically converging methods.

For our final test in this series we again used the trigonometric function just discussed but this time choose parameters so that the solution will be on the nonlinear portion of the curve as shown in Figure III. Now the no leap method for this problem took 435 evaluations to achieve a locally optimal solution of .6706, as compared to 226 evaluations to achieve the global optimum in the near-linear case. The use of leaps yields dramatic improvements in the operation of the code. Table VI displays the results of this investigation. Powell reports solutions to two five dimensional problems with as few as 106 evaluations - which according to our estimates could only be achieved if the function were perfectly quadratic in the region of search. This result emphasizes once more that in nonquadratic cases leaps are useful while in quadratic cases where the ridges are linear they tend to be of little or slightly negative value.

In the next section we explore the effect of our leapsize procedure on some more difficult exponential problems for which the unmodified conjugate directions technique has failed to locate an optimum.

IV EFFECT OF TOLERANCES ON SEARCH PROCEDURE

No technique is impervious to the selection of program parameters. Not only can tolerance parameters affect the length of time taken to converge, but they also can determine whether a particular technique will converge at all. In Table II given earlier we presented the results of our program on the Rosenbrock function. Tolerance characteristics are fairly robust for this problem. Leaps in this

	LEAPSIZE = 1 (POWELL)		LEAPSIZE = 1.5		LEAPSIZE = 2	
ITERATION	NUMBER OF FUNCTION EVALUATIONS	FUNCTION VALUE	NUMBER OF FUNCTION EVALUATIONS	FUNCTION VALUE	NUMBER OF FUNCTION EVALUATIONS	FUNCTION VALUE
1	44	17.154	44	17.154	44	17.154
2	68	7.860	69	13.207	69	15.991
3	92	2.004	94	3.631	94	2.683
4	113	1.967	118	2.191	119	2.459
5	134	1.967	141	1.325	143	2.443
6	159	1.953	165	.216	167	.463
7	182	1.934	188	.042	191	.065
8	206	1.861	211	.001	215	.058
9	231	1.786	233	3×10^{-4}	238	.009
10	252	1.739	253	2×10^{-4}	260	.001
11	275	1.580	274*	2×10^{-4}	282	2×10^{-4}
12	298	1.523			305*	7×10^{-6}
13	321	1.440				
14	342	1.406				
15	369	.943				
16	396	.695				
17	416	.671				
18	435	.671				
19	450*	.671				

TABLE VI - Solution For Contrived Trigonometric Function

*Stopped when no variable changed more than .0001

TOL = 10, .1, .0001
MPASS = 4
MTEST = 1

case also seem to be robust in that they improve performance in all instances.

To study the importance of tolerances in more complex circumstances we used a problem generated by Box (1966). In Box's notation this problem is

Minimize $f(a_1, a_2, a_3) =$

$$\Sigma_x[a_3(e^{-a_1x} - e^{-a_2x}) - (e^{-x} - e^{-10x})]^2$$

where the summation on x is from 0.1 to 1.0 in increments of 0.1. This problem has infinitely many global and local optima. We choose this problem since Box reported that for six of nine starting points Powell's method failed to converge to any of the global optima. Table VII and VIII shows the results for one of these cases, with various parameters. As did Box, we had difficulty getting solutions for any of these problems when no leap was used, although we did find that certain tolerance combinations achieved solutions better than that reported by Box. We also found that in many cases scrapping the conjugate directions from time to time and restarting the search using coordinate directions seemed to improve the rate of convergence. This obviously implies that coordinate directions are useful for this problem. Use of leaps improved convergence in all cases. In some cases convergence to one of the global optima occurred through the use of leaps. In almost all cases using a leap led to an improved value of the objective function. To improve things further still, we suspect that use of a dynamically

TOL			MPASS	LEAPSIZE	NUMBER OF FUNCTION EVALUATIONS	VALUE OF FINAL SOLUTION
1	2	3*				
10	.1	.0001	3	1 (Powell)	251	3×10^{-8}
10	.1	.0001	3	2	118	9×10^{-9}
10	.1	.0001	4	1 (Powell)	179	3×10^{-6}
10	.1	.0001	4	2	140	5×10^{-6}
10	.1	.0001	5	1 (Powell)	178	7×10^{-6}
10	.1	.0001	5	2	153	4×10^{-8}
10	.1	.0001	6	1 (Powell)	187	3×10^{-7}
10	.1	.0001	6	2	162	2×10^{-6}
10	.01	.0001	4	1 (Powell)	313	1×10^{-2}
10	.01	.0001	4	2	191	8×10^{-7}
10	1	.0001	4	1	Found no min in any direction at the 1st iteration using this large a step.	
10	1	.0001	4	2		

* 1 = Upper bound on step size
2 = Step length for first quadratic fit in any direction
3 = Accuracy of linear search

TABLE VII: Results From Three Dimensional (Box, 1966; Start 2) Exponential MTEST = 1

changing leapsize would be advisable. If a leap produced a better solution along some direction we should continue to leap until a worse solution is found before continuing on to the next iteration. Whenever we performed this operation we achieved convergence with Powell's code.

The state of affairs just reported points out a problem with the one-dimensional quadratic fit used both by Powell and by ourselves. The surface seems to be so flat along conjugate directions that the linear search stops far from the optimum along the line. Thus it is difficult to say whether conjugate directions with a different line search routine would not work with this type of problem. At any rate it is encouraging to note that use of leaps not only improves speed of convergence but has the effect of creating a more robust search procedure.

V USEFULNESS OF INTERACTIVE COMPUTING IN THE EXPERIMENTS

The line of research presented in this paper would have been nearly impossible to conduct without using a highly flexible interactive code such as APL. In many ways the study of optimization is typical of highly complex search, evaluation and decision making tasks, where a great deal of information both useful and spurious is available at some cost. Gathering and presenting this information to the decision maker may be at least as difficult as understanding the theoretical and analytical aspects of the problem. The interactive facilities of APL helped in numerous ways in organizing these tasks. First, we found it easy to develop subroutines whose performance we could study in detail prior to including them in our main search routine. Both debugging and operation of these routines could be carried out on an almost instantaneous basis because of the language's interactive capabilities and its ability to

TOL			MPASS	LEAPSIZE	NUMBER OF FUNCTION EVALUATIONS TO ACCURACY OF .0001	VALUE OF FINAL SOLUTIONS
1	2	3 *				
10	.1	.0001	4	1 (Powell)	168	.147
10	.1	.0001	4	2	235	.009
10	.1	.0001	4	4	239	$6x10^{-7}$
10	.1	.0001	4	16	195	$2x10^{-7}$
10	.05	.0001	4	1 (Powell)	147	.149
10	.05	.0001	4	4	145	.149
10	.05	.0001	4	16	103	.009* then diverged
10	.5	.0001	4	1 (Powell)	55	.022
10	.5	.0001	4	2	71	.018

TABLE VIII: Results From Three Dimensional (Box, 1966; Start 2) Exponential MTEST = 1

* 1 = Upper bound on step size
2 = Step length for first quadratic fit in any direction
3 = Accuracy of linear search

monitor in detail the functioning of these subroutines. Second, as through experience we learned which numerical outputs from a given routine were important to monitor, the editing capabilities of APL allowed us rapidly to suppress inessential information. Third, when an hypothesis regarding the operation of some part of the search routine was formulated, the language allowed us instantaneously to test that hypothesis under controlled experimental conditions in which possibly compounding variables could be held constant. Fourth, the ease with which subroutines could be programmed and operated independently allowed us, in effect to develop and utilize tools for conducting these experiments, thus helping us to ascertain certain problem characteristics as our study proceeded. Finally, when hypotheses were shown by controlled experiments to be incorrect, the editing and memory capabilities of the language made it easy to backtrack and hence recover successful approaches before conducting new experiments.

VI CONCLUSIONS

In this paper we considered the performance of Powell's conjugate direction method, as originally developed and as modified in the manner indicated. Very few problems could be found which gave Powell's original technique difficulty in finally converging to a solution. In those cases, restarting the directions or some other interactive parameter change would always start the search moving again - in this sense Powell's technique never failed to converge. However when we introduced the concept of leapsizes, the modification demonstrated a beneficial effect on speed of convergence and the robustness of convergence for the technique in particular for cases where the functions exhibit nonlinear valleys.

It should be said that the concept of leapsizes is not a new one. For instance, Rosenbrock (1960) used this idea. His usage of leaps, however, differed from ours in two ways. First he used leaps with a steepest descent method. Second, he leaped only in an attempt to improve the value of the objective function - not to move away from the ridge itself as we do. This is a very important distinction.

References

Box, H.J. (1966). A Comparison of several methods. The Computer Journal, Vol. 9, p. 67.

Fletcher, R. (1965). Function minimization without evaluating derivatives - a review. The Computer Journal, Vol. 8, pp. 33-41.

Fletcher, R. and Powell, M.J.D. (1963). Rapidly convergent descent methods for minimization. The Computer Journal, Vol. 6, pp. 163-168.

Fletcher, R. and Reeves, C.M. (1964). Function minimization by conjugate gradients. The Computer Journal, Vol. 7, pp. 149-154.

Powell, M.J.D. (1962). An iterative method for finding stationary value of a function of several variables. The Computer Journal, Vol. 5, p. 147.

Powell, M.J.D. (1965). An efficient method for finding the minimum of a function of several variables without derivatives. The Computer Journal, Vol. 7, pp. 155-162.

Rosenbrock, H.H. (1960). An automatic method for finding the greatest and least value of a function. The Computer Journal, Vol. 3, pp. 175-184.

Zangwill, W.I. (1969). Nonlinear Programming: A Unified Approach, Prentice Hall, Inc., Englewood Cliffs, New Jersey, Ch. 6.

APPENDIX: LISTING OF CODE

```
     DESCRIBE
THIS WS EMPLOYS POWELL'S CONJUGATE COORDINATE SEARCH WITH VARIABLE LEAPSIZE,
A PARAMETER DESIGNED TO PERMIT CHANGING THE POINT AT WHICH THE NEXT ITERATION
OF THE SEARCH TECHNIQUE BEGINGS.  THE DIRECTION OF THE LEAP IS THAT OF THE LAST
SUCCESSFUL CONJUGATE COORDINATE SEARCH IN AN ITERATION.
MAIN PROGRAM IS OPERATED USING    ITER,INIT,LEAPSIZE MACR POINT,FNVALUE
GLOBAL PARAMETERS FOR THIS WS ARE MPASS (MAX EVALUATIONS PER PASS),
MTEST (MAXSTEPSIZE REDUCTION BECAUSE OF NO INDICATED MINIMUM)
TOL←MAXSTEPSIZE,STEPSIZE,ALLOWABLE TOLERANCE
COUNT (SET TO ZERO WHEN BEGINNING RUN; GIVES NUMBER OF FUNCTION VALUES COMPUTED
ZMIN GIVES MIN FNVALUE FOUND TO DATE
XX GIVES DIRECTIONS EMPLOYED IN THE CURRENT ITERATION (DELETED DIRECTIONS
MAY BE REPORTED, BUT ARE NOT CALLE DFOR IN THE ROUTINES AS PROGRAMMED).
DATA GIVES HISTORY OF THE SEARCH

      ∇MACR[⎕]∇
    ∇ NTIM MACR INI;DIRE;POIN;W;NJ;REDU;ORT;YY;DIM;DIMM
[1]    ⍝INPUTS ARE NTIM MACR INIT POINT.
[2]    ⍝ TOL MUST ALWAYS BE DEFINED BEFORE RUNNING THIS PROGRAM
[3]    ⍝XX AND DATA MATRICES MUST BE DEFINED IF  NTIM[2]←1
[4]    DIMM←ρINI
[5]    DIM←DIMM-1
[6]    NJ←0
[7]    →ΔQΔ×⍳NTIM[2]=1
[8]    DATA←(1,DIMM)ρINI
[9]    XX←(⍳DIM)∘.=⍳DIM
[10]   DATA←DATA,[1]INI SEEK XX
[11]   NI←0
[12]  ΔQΔ:DATA←DATA,[1](,(¯1,DIMM)↑DATA)SEEK((-DIM),DIM)↑XX
[13]   JM←1↑ρDATA
[14]   DIRE←DIM↓,DATA[JM;]-DATA[JM-NTIM[2]+DIM;]
[15]   DATA←DATA,[1],LMIN(,DATA[JM;]),DIRE
[16]   POIN←DIM↓(NTIM[3]×,(¯1,DIMM)↑DATA)-(NTIM[3]-1)×,DATA[JM;]
[17]   DATA←DATA,[1]POIN,EVAL⍎POIN
[18]   NTIM[2]←1
[19]   YY←((-DIM),DIM)↑XX
[20]   ORT←(YY÷(DIM,1)ρ(+/YY×YY)*0.5)+.×DIRE
[21]   ORT←|ORT
[22]   ORT←((ORT=⌈/ORT)×⍳ρORT)~0
[23]   REDU←(1↑ρXX)ρ1
[24]   REDU[(ρREDU)+ORT-DIM]←0
[25]   XX←REDU⌿XX
[26]   XX←XX,[1]DIRE
[27]   NJ←NJ+1
[28]   (¯2,DIMM)↑DATA
[29]   'ooo   ';'COUNT = ';COUNT;'   ooo'
[30]   →0×⍳NJ>NTIM[1]
[31]   →ΔQΔ
    ∇

      ∇SEEK[⎕]∇
    ∇ RSS←PFV SEEK XX;IM;I;SSMP
[1]    IM←1↑ρXX
[2]    I←1
[3]    RSS←(1,DIMM)ρPFV
[4]   ΔΔΔ:SSMP←,LMIN(,(¯1,DIMM)↑RSS),XX[I;]
[5]    RSS←RSS,[1]SSMP
[6]    I←I+1
[7]    →BBB×⍳I>IM
[8]    →ΔΔΔ
[9]   BBB:RSS←1 0↓RSS
    ∇
```

```
      ∇LMIN[⎕]∇
    ∇ RRR←LMIN X;A;B;C;D;ZA;ZB;ZC;ZD;DEN;ZZ;Z;ZZZ;II;MM;MZZ;STEP;PASS;TEST;PHO
[1]    ⍝USES A QUADRATIC REGRESSION ON  THREE POINTS TO PREDICT A MINIMUM
[2]    ⍝INPUTS ARE STEP[3 COORDS] AND X[POINT,FN VALUE, AND DIRECTION]
[3]    ⍝STEP[1]=MAXSTEPSIZE,STEP[2]=STEPSIZE,STEP[3]=ALLOWABLE TOLERANCE
[4]    STEP←TOL
[5]    PASS←1
[6]    TEST←0
[7]    MM←((⍴X)-1)÷2
[8]   JJ:A←0
[9]    R←2 1⍴0
[10]   II←0
[11]   B←STEP[2]
[12]  J:ZA←X[MM+1]
[13]   ZB←EVAL⍎(MM↑X)+B×(-MM)↑X
[14]   →G×⍳ZB≥ZA
[15]   C←2×STEP[2]
[16]   ZC←EVAL⍎(MM↑X)+C×(-MM)↑X
[17]   R←((R,A,ZA),B,ZB),C,ZC
[18]  I:DEN←((B-C)×ZA)+((C-A)×ZB)+(A-B)×ZC
[19]   D←(((B*2)-C*2)×ZA)+(((C*2)-A*2)×ZB)+((A*2)-B*2)×ZC
[20]   D←(D×0.5)÷DEN
[21]   →H×⍳(|D)>STEP[1]
[22]   →KKK×⍳0≤DEN÷(A-B)×(B-C)×C-A
[23]   MZZ←((|D-A)<STEP[3]),((|D-B)<STEP[3]),(|D-C)<STEP[3]
[24]   →L×⍳(+/MZZ)>0
[25]   PASS←PASS+1
[26]   →L×⍳PASS≥MPASS
[27]  H:ZD←EVAL⍎(MM↑X)+D×(-MM)↑X
[28]   R←R,D,ZD
[29]   ZZ←ZA,ZB,ZC,ZD
[30]   ZZZ←(⌈/ZZ)=ZZ
[31]   →TTT×⍳(+/ZZZ)>1
[32]  TTR:→I×⍳4=+/ZZZ=0 0 0 1
[33]   ZZZ←~ZZZ
[34]   ZZ←ZZZ/ZZ
[35]   Z←ZZZ/A,B,C,D
[36]   ZA←ZZ[1]
[37]   ZB←ZZ[2]
[38]   ZC←ZZ[3]
[39]   A←Z[1]
[40]   B←Z[2]
[41]   C←Z[3]
[42]   →H
[43]  G:C←-STEP[2]
[44]   ZC←EVAL⍎(MM↑X)+C×(-MM)↑X
[45]   R←((R,A,ZA),B,ZB),C,ZC
[46]   →H
[47]  KKK:'TEST INDICATES NO MINIMUM;STEP[2]←STEP[2]÷10'
[48]   TEST←TEST+1
[49]   →L×⍳TEST≥NTEST
[50]   STEP[2]←STEP[2]÷10
[51]   →JJJ
[52]  L:→LL
[53]   'VALUE IS    ';((⌊/ZA,ZB,ZC)=ZA,ZB,ZC)/ZA,ZB,ZC
[54]  LL:PHO←(⌊/ZA,ZB,ZC)=ZA,ZB,ZC
[55]   →LLL
[56]   'AT    ';(MM↑X)+(((PHO×⍳3)=⌈/PHO×⍳3)/A,B,C)×(-MM)↑X
[57]  LLL:
[58]   RRR←(MM↑X)+(((PHO×⍳3)=⌈/PHO×⍳3)/A,B,C)×(-MM)↑X
[59]   RRR←RRR,((PHO×⍳3)=⌈/PHO×⍳3)/ZA,ZB,ZC
[60]   →0
```

```
[61] I:'PREDICTED MIN LARGER THAN KNOWN VALUES; STEP[2]←STEP[2]÷10'
[62]  TEST←TEST+1
[63]  →L×⍳TEST≥MTEST
[64]  STEP[2]←STEP[2]÷10
[65]  →JJJ
[66] N:D←STEP[1]××D-A
[67]  II←II+1
[68]  →NN×⍳II>1
[69]  PASS←PASS+1
[70]  →N
[71] NN:'MAX INCREMENT USED TWICE;MAXSTEPSIZE DOUBLED'
[72]  STEP[1]←STEP[1]×2
[73] JJJ:PHO←(⌊/ZA,ZB,ZC)=ZA,ZB,ZC
[74]  RRR←(MM↑X)+(((PHO×⍳3)=⌈/PHO×⍳3)/A,B,C)×(-MM)↑X
[75]  RRR←RRR,((PHO×⍳3)=⌈/PHO×⍳3)/ZA,ZB,ZC
[76]  RRR←,RRR
[77]  X←RRR,(-MM)↑X
[78]  →JJ
[79] TTT:ZZZ←(⍳4)×ZZZ
[80]  ZZZ←(⌈/ZZZ)=ZZZ
[81]  →TTR
    ∇


      ∇EVAL[⎕]∇
     ∇ R←EVAL X;J;ZZZ
 [1]    ⍝THIS FUNCTION COMPUTES VALUES OF A THREE←DIMENSIONAL EXPONENTIAL.
 [2]    ZZZ←0.1×⍳10
 [3]    R←(⍴X[1;])⍴0
 [4]    J←1
 [5]    R[J]←+/(((*-X[1;J]×ZZZ)-*-X[2;J]×ZZZ)-X[3;J]×(*-ZZZ)-*-10×ZZZ)*2
 [6]    →8×⍳ZMIN≤R[J]
 [7]    ZMIN←R[J]
 [8]    J←J+1
 [9]    COUNT←COUNT+1
 [10]   →0×⍳J>⍴X[1;]
 [11]   →5
     ∇


       ⍝EXAMPLE OF PROGRAM OPERATION

       COUNT,TOL,MPASS,MTEST,BEGIN
  0  10  0.1  0.0001  4  1  2.5  10  10  275.881

       1 0 1 MACR BEGIN
  TEST INDICATES NO MINIMUM;STEP[2]←STEP[2]÷10
  PREDICTED MIN LARGER THAN KNOWN VALUES; STEP[2]←STEP[2]÷10
  TEST INDICATES NO MINIMUM;STEP[2]←STEP[2]÷10
   2.15424382      16.44652126       0.6601690751     0.1639626634
   2.15424382      16.44652126       0.6601690751     0.1639626634
  ooo   COUNT = 22   ooo
  PREDICTED MIN LARGER THAN KNOWN VALUES; STEP[2]←STEP[2]÷10
  TEST INDICATES NO MINIMUM;STEP[2]←STEP[2]÷10
  PREDICTED MIN LARGER THAN KNOWN VALUES; STEP[2]←STEP[2]÷10
   2.105092584     16.30252126       0.6716643134     0.1582622029
   2.105092584     16.30252126       0.6716643134     0.1582622029
  ooo   COUNT = 33   ooo
```

SELECTION AND EVALUATION

Chairman: A.C. Williams, Mobil Oil Corp.

Panelists: J. Gregory Colahan, General Battery Corp.
J.R. Ellison, Mobil Oil Corp.
Milton M. Gutterman, Std. of Indiana
David Hirschfeld, Management Science Systems

Summary of panel discussion by A.C. Williams.

How does an organization select a mathematical programming software system for applications in a commercial environment? What are the relevant factors to be considered, how can they be evaluated, and how should they be weighted in arriving at a conclusion. These are the questions addressed by the panel. On almost all matters, there seemed to be a consensus of opinion, even though the panel members came to the questions from a number of different perspectives. This note is an attempt to summarize that consensus.

As in so many other instances of work-a-day system evaluations, there are three aspects to be considered: (i) the functional, (ii) the technical, and (iii) the economics. These are listed in the approximate order of importance.

Functional concerns, of course, begin with asking whether the particular system under consideration will meet job requirements, both now and in the foreseeable future. This requires an evaluation not only of the system being offered, but of its future possible path of evolvement. And to evaluate that requires that some hard questions be asked about the vendor. Will he remain in business? If he does, will he maintain it and keep it up to date, and is he likely to have it evolve in a direction suitable to our needs?

It is important to realize that the choice of a mathematical programming system is likely to influence and perhaps to influence greatly, the future development of applications in the company. If a code has a good GUB feature, large and comprehensive distribution models can be developed and used. A general purpose MIP feature could allow development in directions which would otherwise not be possible. The same could be said of a quadratic programming option, network flow subroutines, special purpose MIP's, a separable programming option, etc.

An important functional question to be asked of a proposed system is how well it can be integrated into the company's operating procedures. Especially in instances where results are needed on a real time basis, as in some day to day refinery operations, we need satisfactory answers to these questions:
(i). Does the system facilitate getting data and maintaining it?
(ii). Will it be easy to ask questions and get answers (are the matrix generators and the report writers satisfactory?).
(iii), Will the system be reliable?

The question of reliability gets us into the technical questions. Reliability is the question of how bug-free is the code, and how stable is the code, (stability here refers to the code's ability to handle numerical difficulties arising from almost singularities and from degeneracies). It is generally conceded that no code is bug-free or stable for all problems, and that therefore applications systems have to be designed to take those facts into account. A preliminary reading on the reliability of a system can often be obtaining be asking around. The vendor will often supply a list of satisfied customers. However, it should be borne in mind that his list of all customers and his list of satisfied customers may not be identical.

Many of the "old hands" keep a number of test problems, carefully safeguarded from any changes, to be brought forth to exercise a proposed code. Usually, there will be at least one test problem that will upset the stability of any code - the point being to see how long it takes to bring it down, and how gracefully it does down when it finally surrenders.

The test problems should, of course, contain representative problems, typical of those to be solved in production. Most users are not surprised if their repertoire of problems turn up some bugs in a code being tested by them for the first time. The real questions are: how does the vendor respond. Does he give you a work-around? Do new bugs continue to appear, or does the first spate of bugs appear to be all of them?

Bugs and instabilities will persist in every code, and they will have to be dealt with. An important technical property to be looked for therefore, is flexibility to adjust tolerances, conveniently and easily; and to control the flow of the solution, by e.g., adjusting inversion frequency, level of multiple pricing and other parameters.

While the local l.p. technical person wants and needs such controls, in many instances he wants to be able to shield the users from worrying about them. (Some would say the user shouldn't even know about them, but this appears to be an extreme view.) Therefore, the ease with which the system control language allows macros to be written and executed is an important feature in most installations.

In many installations the ability to do recursion would be an important property to be looked for in a code. In this case, it would be important to be able to read solutions, to read the shadow prices, to analyse them, and to revise the coefficients conveniently and easily in a higher level language, such as Fortran or PL/1.

The economics is the matter of cost effectiveness. Benchmark runs have to be designed, run, and analysed. Some of the benchmark test problems will be for the purpose of exercising the code, i.e., testing it for bugs and for its stability. In testing for cost effectiveness, however, actual production conditions should be simulated as nearly as possible. Actual production models should be used, if available. If most production runs are made from an advanced starting basis, with a few revised elements, then the code should be tested under those conditions. The performance of the code in solving a problem starting from scratch is almost irrelevant. If most production runs are made during prime shift in a multiprogrammed environment, then the test of the code should be done during prime shift in a multiprogrammed environment. Of course, results will vary from run to run, so that several runs for each case will generally have to be made to get an average and a variance.

Computation associated with the mathematical programming activities can generally be broken down into (i) matrix generation, (ii) "execution," and (iii) report writing. The "execution" can further be broken down into "primal" and "other" (typically converting the input data into the formats to be used by the code, e.g. SET UP, CONVERT).

For at least one production shop it was reported that the execution step accounts for only about 60% of the total CPU time charged to linear programming, the other 40% being taken up by matrix generation and report writing. And only a total of 40% was spent in primal, i.e., the actual execution of the simplex method calculations. There seemed to be some agreement that these kind of proportions were not atypical. The conclusion is, of course, that the matrix generator and report writers cost performance can be significant and important, and should be included in the benchmark tests.

What to do about tuning is a major headache in benchmarking codes. The amount of main memory assigned can have a major effect on the performance of the code, and, of course, an inverse effect on the other work running in the computer. It is generally recommended that a number of different memory size assignments be tested for the major application types, and that, on the basis of that data, a recommended size be arrived at by use of the charge out system and/or negotiation with the computer center.

How much tuning of the parameters of the code itself should be done is a difficult question. Which parameters are considered most important are, of course, initially best determined by consultation with the vendor or with other users. If the range of different problem types to be solved is fairly narrow, it can pay to spend considerable time and energy to get good parameter settings, whereas if the problem set is very variable and subject to change, it may be that one should not go much beyond using the default settings suggested by the vendor.

Overall, the benchmark problem is

difficult because of the many variables involved, and the fact that run results are themselves variable. Unless great care is taken in designing the tests, the number of runs and the consequent amount of computer resource used can get out of hand. And, unless great care is taken in interpreting and analysing the results, wrong conclusions can be reached.

In summary, any organization with a significant present or contemplated mathematical programming work-load will find it worthwhile to exercise a good deal of care in the selection of a software system. The issues discussed in this note are probably the issues that the organization needs resolved. But, more important than any specific issue is to assign the evaluation task a reasonable priority and budget, and to assign a good person to the task.

- - - - - - - - - -

Attached are tables which may be used as general guidelines to some of the mathematical programming systems and matrix generators/ report writers. The tables are based on the best information that was available at the time they were prepared. New systems and new features are continually being brought out, however, so that any potential user would have to obtain the most up to date information.

NAME	DATAFORM	GAMMA	MAGEN-PDS OMNI	MGRW	Other
TEMPO		X			
APEX III			X		
MPS					CFMS
OPT.SYST.			X		
FMPS		X	X		
MPSX	X	X	X	X	
MPSX/370	X	X	X	X	
MPS III	X	X	X		
LP/System 3			X		

NAME	TEMPO	APEX III	MPS	OPT. SYST.	FMPS	MPSX	MPSX/ 370	MPS III	LP/System 3
COMPUTERS	Burroughs B6700 B7700	CDC Cyber 70 Cyber 170 6000 7600	Honeywell Series 60	NCR Century 200	UNIVAC 1100 Series	IBM 360 370	IBM 370	IBM 360 370	IBM System 3
DATA PACKING		X			X	*		X	
LU ETA FILE with Forrest-Tomlin					X	* +	X	X	
Hellerman-Rarick Invert		X		X		* +	X	X	
DEVEX		X				*+X	X	X	
REDUCE	X	X			X	X	X	X	
GUB	X		X		X	X		X	
MIP	X	X	X		X	X	X	X	
SOS		X	X				X		
SEPARABLE	X		X	X	X	X	X	X	X
CONTROL	On-Line	Call					Call	On-Line	

* WHIZARD Optimizer

+ HOP Optimizer

EXPERIENCES IN THE DEVELOPMENT OF A LARGE SCALE LINEAR PROGRAMMING SYSTEM

Robert J. Sjoquist
Control Data Corporation
4201 North Lexington Avenue
St. Paul, Minnesota 55112

ABSTRACT

There has been increasing interest in software development techniques over the past few years. Effective techniques for developing reliable, maintainable and extendable software significantly reduce total software costs. Several techniques have been described in recent literature. The objective of this paper is to report experience with some of these techniques in the development of APEX-III, a large scale linear programming system.

Several of the techniques are felt to contribute to a successful development. They include a project staff organization similar to a "chief programmer team", documentation and coding standards, and open programming.

Techniques which did not appear directly applicable to the development were structured programming and top-down programming.

Finally, it is noted that a significant amount of effort is required to prepare accurate estimates of development costs.

INTRODUCTION

Last year, our company completed development of a large scale linear programming system, APEX-III. Several recently publicized mathematical programming and software development techniques were considered and used in the development. Over the past year, we have had the opportunity to evaluate the effectiveness of these techniques.

A development of this kind requires two distinct types of skills: mathematical programming skills and software development skills. By mathematical programming skills, we mean those skills unique to a linear programming system. They include a knowledge of the type of problems to be solved. An efficient design considers model characteristics such as size, density and structure. Familiarity with characteristics of current algorithms is needed to properly match the algorithms to the computer. Performance and numerical stability of a code reflect the programmer's understanding of the algorithm.

Just as important as the mathematical programming skills, are the software development skills. These are the skills required for development of any large programming system. They include a knowledge of the computer, the operating system, the compilers and assemblers. They include a familiarity with data handling techniques and means for efficient use of the hardware. These skills also include a working knowledge of techniques for producing reliable and maintainable software products.

The purpose of this paper is to report our experience with these software development techniques.

THE DEVELOPMENT PROJECT

The development occurred over a four-year period and in two distinct phases. The first was the development of an in-core, no frills, linear programming system. Five programmers completed the development in less than one year The code was refined over a two-year period by one programmer working on a casual basis.

The second development phase involved adding several major features to the system. These included an out-of-core capability, mixed integer and parametric programming, and a matrix reduction facility. The product was required to operate on three series of computers and run under three different operating systems. This development phase involved as many as 12 programmers over a 1-1/2 year period. The objective of each phase was to produce a commercially profitable software product.

THE DEVELOPMENT TECHNIQUES

Project staff organization was similar to the "chief programmer team" described by Baker and Mills {1}. The chief programmer team consists of a chief programmer, a backup programmer, a project secretary, and a pool of programmers.

The chief programmer is generally the most senior member of the development team. He is a working member of the team with overall technical responsibility for the project. On this project, his duties included system design, assigning tasks to project members, and co-ordinating efforts of individual programmers. He also participated in coding and testing the system. In the first phase of development, he produced as much code as any other project member. In the second phase, he produced far less code. More time was spent on design and technical administration of the project.

The backup programmer assists the chief programmer and shares his duties. He is familiar with the entire project and is able to assume chief programmer responsibility at any time.

The project secretary maintains a master library of the system and documentation during development. The secretary incorporates new and corrective code in the system, keeps a record of all changes made, and maintains listings of the system. The project secretary described by Baker and Mills also was responsibile for making and keeping a record of all test runs. In this development, project members took turns handling the secretarial duties. Only the most significant test runs were recorded during the development.

System subroutines were written in FORTRAN and assembly language. Routines critical to performance and those handling packed data were written in assembly language. Remaining routines were written in FORTRAN. Extensive use of FORTRAN eased the problems of executing under different operating systems. Subroutines could be coded and tested more quickly using FORTRAN than using assembly language. This left more time for writing performance critical routines.

Documentation standards were specified and enforced. At the start of the project, each member was given a loose leaf notebook for internal documentation. The notebook was added to as subroutines were defined and designed and updated as routines were modified. Internal documentation consisted of subroutine descriptions, communication region {CR} cell descriptions, and array/file formats.

Subroutine and CR cell descriptions were maintained on the program library. This documentation was extracted from the library via computer program, copied and distributed to project members whenever changes were made.

Subroutine documentation consisted of comment cards within each routine. We felt documentation was more likely to be kept current when included in the routine. Documentation was then updated at the same time and in the same manner as the executable code. Two levels of documentation were required for subroutines. The first level describes what the routine does and how it is called. This level is written when a necessary operation is defined. The second level is a step by step description of how the routine performs its function. This documentation is the result of subroutine design and is equivalent to flow charts. Both levels are written before the subroutine is coded.

CR cell definitions and documentation were maintained as a data file on the system library. Subroutine COMMON statements were generated from this file by a computer program. This program also produced printed documentation of the CR cells.

A minimal set of coding standards were established. The primary pupose of these standards was to promote simple, readable code. The standards were intentionally minimal so they were easily followed. Standard FORTRAN calling sequences were used for all subroutines. Subroutine names begin with a Q or a J if a function subroutine. CR cells were named to differentiate them from local variables and to describe their use. The first character of a CR cell name specifies whether its contents is alpha, integer, or real. The second character of the name specifies its use. The name then defines the cell to be a name, switch, index, count, parameter, or tolerance. Statement numbers in FORTRAN routines are in ascending order. Each routine was restricted to a single exit.

Structured programming, top-down programming, and open programming were considered for the second development phase.

Structured programming {1,2,3} generally does not allow GO TO statements. Rather, all decisions are programmed using IF-THEN-ELSE and DO constructs. The purpose is to provide easily readable code by avoiding complicated branches of control. The IF-THEN-ELSE constructs serve to define all conditions under which a section of code is executed.

One routine was written using structured programming to evaluate its usefulness for this development. The routine performed a complicated vector packing operation.

Structured programming rules were rigorously applied for the investigation. The technique did not produce the desired results. Rather, the structure had to be forced upon the routine and detracted from its clarity and performance. The technique was not used in remaining development. However, the idea was not entirely rejected. Reasonable efforts were made to avoid unnecessary backtracking and to allow control to flow from top to bottom. Sections of code which could be entered in special ways were flagged with comment cards.

Top-down programming {1,3} is a technique to organize the development into levels of detail. The basic inputs, outputs, and operation of the system are first defined. Next a first level of detail of operation is added to the system. The process is repeated until all levels of detail are defined. Each level can be designed, coded, and tested independently of succeeding levels. The technique was not used in this development. We felt that design of an efficient linear programming system consists of several iterations a top-down, bottom-up sequence. Performance critical routines are dependent on details of method and file and array formats. These aspects need to be formulated quite early in the design process. To a degree, these details determine the form of higher level routines.

Open programming {1,2} is a technique which was used in this development. With this technique, each programmer's design, documentation, and code is reviewed by another project member. A thorough review should result in improved readability of documentation and code. The review should also uncover errors that may be missed during system testing. A thorough review also requires a considerable amount of effort. In this development, all documentation and approximately half the code was reviewed. Each programmer first had design documentation reviewed by another programmer. After review and coding through a clean compile, code was similarly reviewed. Additionally, all design documentation was reviewed by either the chief or backup programmer.

Software verification is a difficult and costly part of any development. It is economically infeasible, if not impossible to find all errors in a software product. Nevertheless, the software must work correctly nearly all the time to be useful and profitable. Nearly all software development techniques described above contribute to software reliability. Three direct methods of verification were used in this development.

The open programming review was the first means of verification. Secondly, several utility routines were tested independently of the system in a simulated environment.

This was an attempt at checking out all paths through critical routines. Thirdly, the integrated system was tested. Several models were generated to test special conditions. The bulk of the testing, however, relied on real world linear programming models for system checkout.

CONCLUSIONS

The project was completed on schedule, costs were within 10% of budget and performance and reliability standards were met.

Documentation and coding standards were considered critical to success of the project. Without these standards, it would not have been possible to integrate and debug the system on schedule. Additionally, documentation produced during the first phase of development considerably eased the second phase effort. Several of the problems that were encountered were in areas where standards were not closely followed.

Open programming appeared to be well worthwhile. The review uncovered bugs and resulted in improved documentation. We feel the time spent on the review was more than made up for in the test phase. Initial debugging indicated reviewed code had approximately 1/3 the errors of non-reviewed code. The quality of review could be improved if all done by the chief or backup programmer. These two individuals are most familiar with the entire system. However, this places additional time demands on two critical resources.

Project statistics confirmed the theory that considerable effort is required to produce accurate estimates. The project allowed spending a portion of the original budget before producing final cost estimates. Table 1 compares the accuracy of these estimates to the portion of work complete when estimates were made.

Table 1
Errors in Estimation of Major Tasks

Task	Portion Complete Before Estimate	Error in Estimate
A	19%	under 26%
B	12%	nil
C	3%	under 80%
D	17%	over 16%

The error in estimate for task A was caused by a change of scope imposed upon the project. The negligible error for task B shows that costs can be controlled, given a reasonable estimate. The error for Task C resulted because insufficient time was spent on the estimate. Task D was completed with less effort than estimated because a simpler implementation was devised.

Finally, it should be mentioned that individual programmers' attitudes are key to the success of any development. Projects are completed on time by programmers who are determined to finish on time. Quality products are turned out by programmers who take pride in the quality of their work. These programmers can make effective use of new software development techniques.

REFERENCES

1. F. T. Baker and H. D. Mills, "Chief Programmer Teams", DATAMATION, December, 1973

2. J. R. Donaldson, "Structural Programming", DATAMATION, December, 1973

3. E. F. Miller and G. E. Lindamood, "Structured Programming: Top-Down Approach", DATAMATION, December, 1973

MATH PROGRAMMING USERS VS THE COMPUTER CENTER
(A personal perspective as seen from a foxhole)

J. R. Ellison
Mobil Oil Corp.

The fact that there is a general and ongoing conflict between the users and their computer center is generally, and I believe correctly, taken for granted. If you will accept this as a fact of life, let's analyze why this conflict exists.

The first, and I believe foremost, reason is that each views the other as the proverbial "black box" and, since there is little personal contact with or real understanding of the other's problems or motivations, conflicts arise.

One source of conflict is the difference in objectives. The primary objective of a computer center is often stated to be to "maximize" throughput, efficiency, etc. This can be measured as computer center income (charges) vs budget prediction, as how well job schedules are met, or as wall clock hours or CPU hours per day the center runs. Other popular measures are how many jobs are run each day and "percent CPU utilization".

The Math Programming user usually has as a primary objective the provision of an answer to a question asked by some level of management pertaining to the economic consequences of taking or not taking some course of action. This question includes defining the physical courses of action which go with the economics. In providing the above answer, or answers, the Math Programming user is seldom able to schedule his work since questions are addressed to him and "outside" data is provided to him on a time table which he cannot control and the answers are required by a predetermined date (preferably yesterday) which is not under his control and is often not under the control of the person asking the question. The resulting work load disrupts the computer center schedules with a sudden onslaught of computer load as measured by CPU time without a corresponding increase in the number of jobs run.

The user's habit of preferably working only days (prime shift) compounds the problem since he wants to make and correct errors on the same day so he can submit more work that night or, better yet, get more prime shift runs that day. The Math Programming user is also "problem" oriented and looks on the computer center as the provider of a tool required to solve "his" problem so that he can provide an answer to a complex question. He views the center as a cost and as an obstacle to be overcome when it does not meet his requirements. After all, computer centers don't make money - only answers make money.

Comparisons of the above statements show areas of conflict between scheduled and unscheduled work, specific vs broad objectives, and differences in objectives and measurement of effectiveness.

The computer center viewpoint can often be characterized as a responsibility for maintaining a complex operating system and hardware which must respond to the demands of a wide variety of users, none of whom understand the computer center's problems or how a computer actually functions. They also appear to believe that Math Programming users are unreasonable people who can shut down or stretch out job processing for other users (especially scheduled users) if they are allowed the amount of CPU time they need. Unfortunately, these viewpoints often have a large element of truth in them. Stretching out of job processing causes scheduled jobs to miss their schedule and has been known to make accounting type users extremely unhappy with the computer center. This is important to the computer center because accounting types are much more numerous, are very adept and well trained at writing complaining memorandums, and the computer centers are often controlled by accounting types. If you don't like the above, just substitute technical types or whoever has been causing you the most trouble and it will be just as applicable.

The Math Programmer's point of view is that "we" have to obtain "optimal" answers to questions posed by management and these answers must be available by a given deadline. Unfortunately, we must expend that limited resource known as time to obtain data necessary to understand and correctly model the problem as stated; we must spend additional time removing "bugs" from the model and revising the model to fit the actual question as opposed to what we thought the question was and, finally, we must obtain sufficient time from the computer center to allow us to meet our answer deadline and thus stay out of trouble. This last item is known as providing "reasonable" turnaround and, since it is at the end of the time chain, is the source of much agony to all concerned.

The Math Programming user, when asked by the center how much computer time he will need, simply says he doesn't know. If he knew how much time he needed and how many runs, he would know the answer and would not need any. While there is much truth in this, most computer centers still like predictable and well scheduled runs which he has just told them he doesn't have.

Now, with the above problems and conflicts of interest, how can we learn to co-exist with a computer center and get work accomplished without giving in to them. I believe that it is possible by defining your general requirements in terms of the following:

(1) CPU time required for prime shift and total shift.
(2) Job turnaround. Analyze what you are asking for and be reasonable. Turnaround isn't as simple as it sounds.
(3) Storage space requirements. Permanent storage space is critical. Disk space is almost always in short supply, expensive, and highly desirable. Tape is plentiful, cheap, and less desirable since it must be mounted. System "scratch" space (temporary space) is also limited since it is usually disk. Remember that tapes can be limited by the number of tape drives available. You must be realistic with your space demands if you expect turnaround since multiprogramming means that there are more users than just you demanding space at any one time. Extreme requirements can delay your jobs.
(4) Memory requirements - memory can be expensive and limiting. Be reasonable and careful even if VS (Virtual Systems) is around.
(5) Math Programming codes - you should know more about which codes are best for you than the computer center does, but be prepared to prove your point of view and to justify the costs.

To define the above requirements in anything approaching a quantitative manner, you need tools which will allow you to define where you are and where you were. Reports based on the Systems Management Function (SMF) or its equivalent are the best that I have found. Don't rely on opinions, feelings, "informed" opinions, etc. They are unreliable, cannot be compared, and are an excellent booby trap for the unwary. Careful hand tabulation of data from your runs is better than nothing but very time consuming, especially if you collect enough data to be really useful.

Now let's touch on some of my personal biases and prejudices (I prefer to call them measurements) which I swear by when they agree with my point of view and at when they disagree.

(1) User turnaround - time measured from job in at the reader to finished job output available to the user. The user correctly loves this but be careful. Printout priorities and length of printout (lines and printer speed) can influence this measure and, since this isn't truly controlled by the center, is not exactly a fair measure of computer center performance.
(2) Center turnaround - time measured from job in at the reader to job finished by the initiator and turned over to the output queue. I consider this a measure of the center's ability to provide service. It includes time in the input queue plus time in the computer and thus measures items "controlled" by the center and is a measure of their ability to do work.
(3) Execution ratio - wall clock time from initiator start to initiator finished divided by problem program CPU time. This is an indication of the load on the CPU and can be used as an indication of whether or not you are getting your "fair" share of the computer. It is also useful in estimating how long your job will be in the computer.

The above three measurements must be accumulated, averaged over a reasonable time span, and then plotted so that you can see trends or changes from whatever "normal" is or becomes.

Additional information which allows you to see how your job load fits into the computer center and its schedule are helpful. Data on what percent of the total CPU time, charges, or jobs you represent can be useful especially if it is significantly large. You should also be aware of any special services the center does, or doesn't, perform for you.

Once you have all of this information, you should meet with computer center management and try to obtain commitments and agreements on levels of service which they will provide to you. This written agreement will then provide a basis for co-existence between you.

OPERATIONAL MANAGEMENT OF MATHEMATICAL PROGRAMMING BASED PLANNING SYSTEMS

E. G. Kammerer
UNION CARBIDE
CHEMICALS AND PLASTICS DIVISION
Operations Planning Department

Chemicals and Plastics productions account for approximately 40 percent of Union Carbide's sales resulting in the movement of 19 billion pounds at production annually. This activity involves more than 6,000 products manufactured at approximately 15 plants and moved through more than 100 bulk terminals and warehouses. We deal with more than 30,000 industrial customers utilizing six deepwater vessels, 150 inland barges, and more than 8,000 tank cars, hopper cars, and van boxes.

My purpose in describing Union Carbide's production/distribution network is to highlight the need for the use of mathematical optimization techniques in effective planning. Obviously, it is a complex business, with highly intergrated product/location characteristics in a changing business environment.

The planning system used to help manage our chemical operation generally consists of three phases:

1. Gathering and processing data as input to the mathematical optimization system.
2. The solution of a mathematical model to develop the optimal production and distribution plan for the period.
3. Communication of the results of the planning model to the various functions responsible for the operation.

(See attached flowchart of planning cycle.)

The successful management of a planning system depends on all three of these phases. Any one without the other two is useless. I will discuss each of the phases with particular attention to the needs of the mathematical model.

The first phase of the planning system, the gathering of data, can be broken in two separate activities. The first is standard data or that information which does not change very rapidly. Normally, a yearly review and update of this is sufficient. The type of data included here is annually budgeted cost, capacities, production factors, etc.

The other class of input data consists of the more current operating data which can change from day to day. Transportation cost and limitations, inventory strategies, production limitations and strategies, and the sales forecast are examples of this type of input.

This strategic information is the more difficult and the most critical of the data need. It is gathered through direct interface with various operating groups. A good understanding of their functions and objectives is necessary so that the input can be interpreted. Normally, people in operations are not knowledgeable in the functioning of mathematically based planning systems. Because of this, all meetings to gather this input data must be handled with the idea of getting the understanding and cooperation of these various other groups. It is my responsibility to interpret their input and strategies and develop the proper model representation. This must be done without putting unnecessary restrictions in the model which will prevent true optimization.

The second phase of this process is the solution of a mathematical model. The particular model used by Union Carbide's Chemical and Olefin Division consists of a linear program matrix containing approximately 12,000 variables and 3,000 equations. Variables represent all raw materials, intermediates, and finished products. A number of non-linear plant models are linked to this linear matrix. The resulting model is solved using a recursive technique on an IBM System/370 Model 168, Mathematical Programming System Extended (MPSX) is used to set up and solve the linear portion. Depending on the number of iterations required to reach optimum, the computing time required is normally less than two hours. It is important to note that this is a batch type system. The problems involved with running a batch system are quite different than an on-line system.

In running this model, we interface with two separate functions. Computer Operations has responsibility for hardware availability and priority. Systems Support is responsible for supplying programming, system maintenance, or software support if needed. Problems often develop in obtaining necessary priority due to the size of the planning system and the keen competition for limited computer resources. Because of this, it is desirable to place model runs on production status. That is, runs should be planned in advance and the support commitment of Computer Operations obtained.

The demand on computer resources is cyclic. During particular times of the month or quarter, the demand on the computer is particularly heavy. By planning your major runs during low demand time or off-hours such as nights and weekends, it is possible to improve your turnaround time.

Anyone involved in programming should be aware of the importance of program efficiency. The CPU time required by a job affects what turnaround it will get. Jobs that stay in the computer and slow down the network are not run during high demand time. When running on virtual storage, as is the case in most large installations, the efficiency of the program, that is, the ration of CPU time to elapsed time, is extremely important. The longer a job stays in the computer, the greater the chance of system problems developing and the premature cancellation of the job. In addition, the operating cost of running the system will be reduced. Various techniques can and should be used to improve program efficiency when designing a batch type system that will be operated regularly.

Another important consideration is to insure that any important instruction to the operator, such as mounting tapes or writing over tapes, should be given as close to the time the action is to be taken as possible. The manager of the planning activity has little control over the terminal operators. In addition, the terminal operators are involved in many activities at the same time. Therefore, it is necessary to minimize the opportunity for misunderstandings that may result in premature termination of a run. In large systems, this can be extremely costly in terms of cost and time.

At the present time, we are operating our system at a remote terminal. This arrangement adds to the difficulties mentioned above. It is impossible to be as familiar with changes in software and hardware as when operating directly from the computer center. In the past, we were located at the computer center and a better understanding and effective use of the computer was much easier.

When developing and programming a system such as the one I have described, that is, one that will be run routinely in support of a planning or operating function, it is very important to involve the system analyst when deciding such elements as the number of variables, procedures for updating, maintenance, etc. It seems that time and time again we are in a situation where we need to be able to analyze one more variable than the program will allow and the limit on variables was purely an arbitrary decision made by the designer. When deciding the limit of program capability, the operating analyst is in a better position to determine the need for more or less flexibility.

A very important part of the planning system is the report writer. While the system analyst may be able to interpret a L.P. matrix, the need to review the preliminary results with other functions exist. This highlights the need for reports that can be used to review the solution and identify needed corrections. These reports must be readable by persons not familiar with computer output. The design of these preliminary reports can have a great affect on the ease of review and the reasonableness of the final plan.

The third and final stage of the total planning system is the communication of the results. As I stated earlier, all elements are equally important. The best designed program with the best input is useless if nothing is done with the results.

I am generalizing when I say this but it is my personal feeling that efforts to use direct computer output as the means of communicating the results to the various functional groups are not as effective as interpretation by an analyst. We do use computer output when detail is needed for specific action. But I do not feel it can interpret and communicate the results of such a complex planning system. A considerable effort on the part of system analysts and other experienced people is needed to interpret the results and explain all of these significant implications. A knowledge of the products and the distribution network is as important as computer experience.

Your reaction at this point may be that mathematical programming plays no part in the last phase. This is not the case. The understanding and cooperation of the programmer is essential to the system analyst when evaluating the L.P. solution. I have found that the more significant elements of the solution are not straightforward. Usually, some follow-up case studies or sensitivity analysis is needed. The mathematical programmer plays an important role during these studies. His advice and understanding of the program logic can save considerable time and result in a more meaningful plan.

I would like to restate my position on the final stage. It is my opinion that direct output from the computer is not the best mode of communicating the optimal solution. It is more important that the report writer capability support the efforts of a systems analyst in interpreting the results.

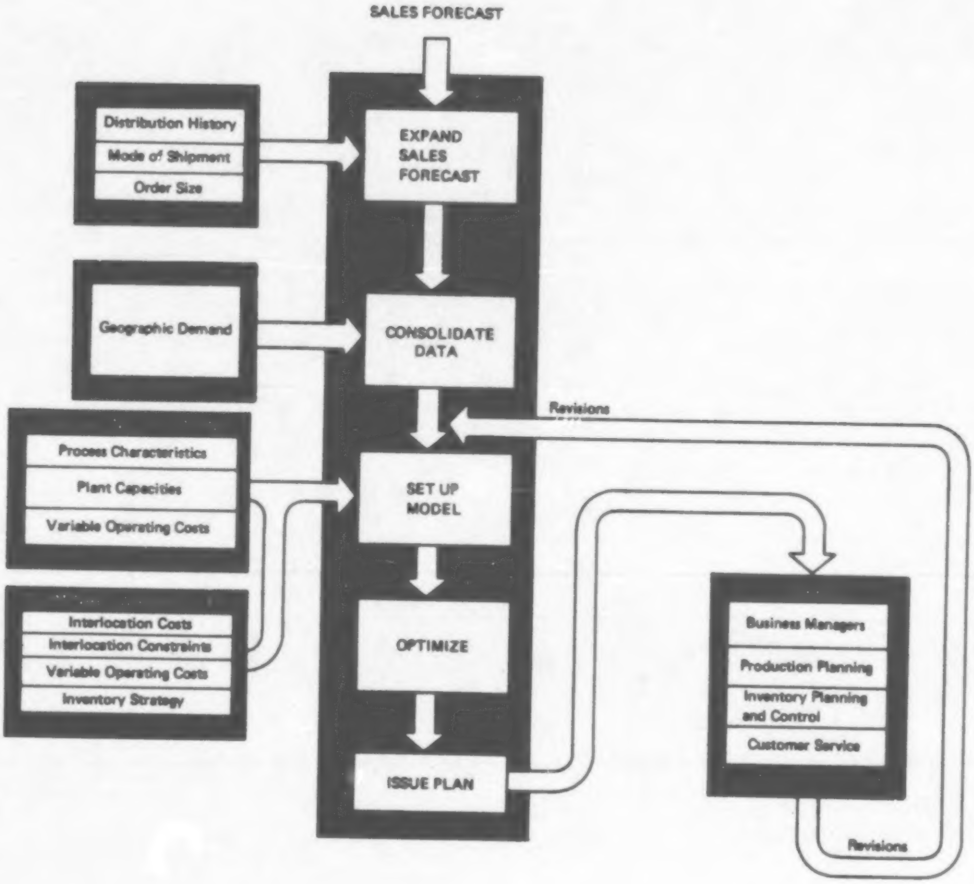
Communication of the plan can take various forms depending on the audience and nature of the activity itself. Normally, greater understanding is obtained through graphs, tables, and diagrams. Prose should be used to elaborate the results. A clear statement of all assumptions and significant limits should be made.

One of the most important points I would like to communicate is the type of result generally looked for. I hope that an understanding of this will help the mathematical programmer to do a better job in using program logic. The most important part of the solution is the direction a business activity should take and not the exact numbers contained in the solution. The business environment changes so rapidly that it is nearly impossible to pin down exact values. A good understanding of the direction or strategy that should be applied in critical areas such as production, inventory, distribution ,etc., is essential. Programming efforts should be done with greater understanding of this fact.

My objective during this presentation has been to give you more insight into the problems and techniques of running a large mathematically programmed planning system. I hope that this knowledge will help you or give you a better feel for some of the critical elements in programs supporting large operating systems. I'm sure that a slightly different viewpoint would be presented if the system under discussion was an on-line system. I do not have much experience with on-line systems and you should keep in mind that all of my discussion is based on batch type systems.

I support your efforts in trying to create better understanding between system designers and system operators. I'm sure that all of us will be able to do a better job if we apply some of the experience gained through understanding the other person's problems.

PLANNING CYCLE


SALES FORECAST
Distribution History
Mode of Shipment
Order Size
EXPAND SALES FORECAST
Geographic Demand
CONSOLIDATE DATA
Revisions
Process Characteristics
Plant Capacities
Variable Operating Costs
SET UP MODEL
Interlocation Costs
Interlocation Constraints
Variable Operating Costs
Inventory Strategy
OPTIMIZE
ISSUE PLAN
Business Managers
Production Planning
Inventory Planning and Control
Customer Service
Revisions

MANAGING A LARGE SCALE PRODUCTION AND DISTRIBUTION SCHEDULING SYSTEM

Kenneth Goldfisher
Nabisco, Inc.

The Biscuit Division of the Nabisco Corporation presently has 16 bakeries, each capable of producing some, but not all, of approximately 750 different products. Shipping branches perform the warehouse and distribution functions required after the bakery has produced and packaged the products. They supply products to local sales branches, of which there are approximately 225 located throughout the United States.

Estimates of sales by product, over a 12 week horizon, are made at each sales branch. These are sent by telecommunication equipment to a central location where they are converted into production requirements which are then allocated to individual production facilities, i.e., ovens, incing machines and packing equipment. The resulting production plan specifies the amount of each product to produce on each facility for each sales branch during each of the four-week periods in the horizon. The production plan minimizes the total variable cost of production and distribution while insuring its production feasibility.

This paper concentrates on the operational aspects of the system, the use of the computer and its relationship to the real world, both from the field unit and the division management point of view.

NBS-114A (REV. 7-73)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET**	1. PUBLICATION OR REPORT NO. NBS Spec. Publ. 502	2. Gov't Accession No.	3. Recipient's Accession No.

4. TITLE AND SUBTITLE

Computers and Mathematical Programming

5. Publication Date
February 1978

6. Performing Organization Code

7. AUTHOR(S)
William W. White, Editor

8. Performing Organ. Report No.

9. PERFORMING ORGANIZATION NAME AND ADDRESS

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

10. Project/Task/Work Unit No.

11. Contract/Grant No.

12. Sponsoring Organization Name and Complete Address *(Street, City, State, ZIP)*
Special Interest Group on Mathematical Programming
Association of Computing Machinery
1133 Avenue of Americas
New York, New York 10036

13. Type of Report & Period Covered
Final

14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 77-600065

16. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)*

The Bicentennial Conference on Mathematical Programming, held in Gaithersburg on November 29 - December 1, 1976, examined the relationship between mathematical programming and the computer. The more than 50 papers and panel discussions exhibited this theme in terms of the design for, use of, implementation of, and implications for mathematical programming software and computations. Particular emphasis was placed on bringing out computer-oriented subject matter not ordinarily presented in a mathematical programming context. These resulting Proceedings document this Conference, which was jointly sponsored by SIGMAP of the ACM and by the Applied Mathematics Division of the Institute for Basic Standards for NBS.

17. KEY WORDS *(six to twelve entries; alphabetical order; capitalize only the first letter of the first key word unless a proper name; separated by semicolons)* Algorithm evaluation/ computer science; computer software; databases; linear programming; management science; mathematical programming; mathematical programming education; nonlinear programming; operations research; software development.

18. AVAILABILITY ☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Sup. of Doc., U.S. Government Printing Office
Washington, D.C. 20402, SD Cat. No. C13 . 10: 502

☐ Order From National Technical Information Service (NTIS)
Springfield, Virginia 22151

19. SECURITY CLASS (THIS REPORT)
UNCLASSIFIED

20. SECURITY CLASS (THIS PAGE)
UNCLASSIFIED

21. NO. OF PAGES
383

22. Price
$5.50

USCOMM-DC 29042-P74